Approximate Counting

Gil Lopes Teixeira

***Resumo* – Este relatório apresenta dois métodos não determinísticos de contagem de caracteres em texto e compara-os com um Contador deterministico. O primeiro método apresentado apenas efetua uma contagem com probabilidade, fixa, de 25%. O segundo é um Contador com probabilidade descrescente com um fator:**

$$\frac{1}{\sqrt{3}^k}$$

Sendo k o total de contagens já efetuadas.

***Abstract* – This report presents two non deterministic methods to count characters in text and compares them to a deterministic counter. The first method is a fixed probability counter that counts every character with the same probability of 25%. The second method is a decreasing probability counter with a lesser factor of:**

$$\frac{1}{\sqrt{3}^k}$$

k being the total amount of times it already counted.

I. Introduction

Non deterministic counters are very useful when working with big data. They produce smaller data structures that still contain most of the information and that are very useful for statistic analysis. All the tables shown here are excerpts from excel files available in this reports folders that contain statistics from 1000 runs of each counter for each language.

II. Fixed Probability Counter

This counter counts with a, fixed, probablity of 25%. After a formal analisys of the algorithm we will compare the number of comparations, effective counts and expected counts.

```
def count_fixed_prob(self, fp, prob=0.25):
    index = {}
    for line in fp.readlines():
        for word in line.split(' '):
            for char in word:
                to_count = random.random()
                if to_count <= prob:
                    if char in index.keys():
                        index[char] += 1
                    else:
                        index[char] = 1
    return index
```

For a given file with n characters excluding spaces the algorithm does:

$$n + k$$

k being the number of times the counter counted. Since the probability is fixed at ¼:

$$k \approx \frac{n}{4}$$

So the expected number of comparisons is:

$$\frac{5}{4} * n$$

char	exact_val	expected
E	14765	14762
T	9214	9211
O	9121	9122
A	7508	7507
H	6886	6889
S	6864	6862
I	6680	6670
N	6455	6456
R	6171	6173
L	4625	4624

Top 10 characters in the english version of the book from Shakespears book Winter Tales.

The expected count was calculated by multiplying the average count of each character, in 1000 tests, by 4. For all the characters with more than 1000 occurrences in the text the relative error was always bellow 0.15%!

French version:

char	exact_val	expected
E	23186	23196
S	11874	11874
I	10429	10427
N	10390	10395
U	10053	10052
A	9958	9969
R	9904	9909
O	9465	9462
T	9438	9439
L	8093	8096

Although the order of the most frequent charactes is diferente they are mostly the same for english and french.

For the french text 185595 comparisons happened and the expected value was: 185823!

III. Decreasing Probability Counter

This counter keeps a table with probabilities for each character. At first they all start with probability 1 and then keep being multiplied by the factor each time that character is counted.

```
def count_decreasing_prob(self, fp, prob_decrease):
    index = {}
    probs = {}
    for line in fp.readlines():
        for word in line.split(' '):
            for char in word:
                if char not in probs:
                    probs[char] = 1
                to_count = random.random()
                if to_count <= probs[char]:
                    if char in index.keys():
                        index[char] += 1
                    else:
                        index[char] = 1
                    probs[char] *= prob_decrease
    return index
```

For a given file with n characters excluding spaces the algorithm does:

$$n + k$$

k being the number of times the counter counted. Since the probability decreases with a factor of $\sqrt{3}$:

$$k \approx \frac{\sqrt{3}^n - \sqrt{3} + 1}{\sqrt{3} - 1}$$

This is the way we estimate expected counts with this counter!

char	exact_val	expected
E	14765	13309.185509071787
O	9121	7768.5864713221035
T	9214	7312.333986458101
A	7508	6409.391194556587
S	6864	6167.604698086741
H	6886	6066.518818206693
I	6680	5999.583573421255
N	6455	5679.933628935697
R	6171	5115.765137172723
L	4625	3823.5051051926444

Top 10 characters in the english version of the book from Shakespears book Winter Tales.

The expected count was calculated by using the average count of each character, in 1000 tests, as k in the following formula:

$$\frac{(a^k - a + 1)}{(a - 1)}$$

The relative error averaged 13% as the example used was small in order to test more times.

_key	exact_val	expected
E	23186	18102.568650951383
S	11874	10394.087297395796
A	9958	10056.67902266104
I	10429	9364.104142942328
T	9438	8765.785016084745
N	10390	8575.907484958707
R	9904	8388.762004640239
U	10053	8297.238302586682
O	9465	7853.280046356739
L	8093	7474.891009508449

The same holds for this counter. The top 10 letters are mainly the same though the order is different.

Number of comparisons: 149098.

Expected comparisons: 148680!

IV. Conclusion

The fixed probability counter works very well for small and medium sized examples but for large examples the decreasing probability counter behaves best! Resulting in less comparissons and reasonable error for large data sets. The counters top characters seem to be the same set for either French or English in the same literature. In order to further confirm it the decreasing probability counter was also ran on an English and Swedish versions of the odyssey. It confirms that as the size of the document increases, in this case about 5x, the relative error tends to decrease. If you wish to see stastistics on the data presented or the odyssey example you should refer to the excel files included. In there you have per character the maximum, minimum and average counts of the 1000 runs. Furthermore it includes the absolute and relative errors for each character.