Most Frequent Words

Gil Lopes Teixeira

***Resumo* – Este relatório apresenta um método de estimar a frequência de palavras chamado LossyCounting. Este método recebe um parâmetro k e um texto de tamanho n que permite manter apenas as palavras que no total tenham $\geq \frac{n}{k}$ no seu contador e que portanto aproxima os resultados por cima com um erro máximo de $\frac{n}{k}$.**

***Abstract* – This report presents a method to estimate the frequency of words called LossyCounting. This method receives a parameter k and a text of length n that allows to only keep words with $\geq \frac{n}{k}$ in it's counter and it never understimates the results, overestimating at most $\frac{n}{k}$.**

I. INTRODUCTION

The Lossy Counters allows us to have an estimate of words frequency. The difference between this and exact counting is that in this method we discard words with under $\frac{n}{k}$ counts. 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. LOSSYCOUNTER

```
def LossyCount(k):
    T = {}
    n = 0
    Δ = 0
    for line in fp.readlines():
      for word in line.split(' '):
        n += 1
        if word in index.keys():
          T[word] += 1
        else:
          T[word] = 1 + Δ
        if math.floor(n/k) != Δ:
          delta = math.floor(n/k)
          for word in T:
           if T[word] < Δ:
             T.pop(word)
    return T
```

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

$$\sum_{i=0}^{n-1} 1 + 1 + delete(ci < \frac{i}{k})$$

The delete happens every iteraction since delta, an implicit value, is updated and compared every iteraction to $\left\lfloor \frac{i}{k} \right\rfloor$ and runs over the current index.
This algorithm might overestimate de result by the n/k but never underestimated! This means that for a large n the algorithm behaves badly for small k!

III. RESULTS

For testing two different books were used. Winters tails in English, with 26880 words, and French by Shakespear, and The Odissey by Homero in English, with 133590 words, and Swedish.

III.1 WINTERS TAILS - SHAKESPEAR

For the first example, in English, running the lossy counter with k=1000 we know that at most the frequency will be overestimated by no more than $\lfloor 26.88 \rfloor$ as it's confirmed in the table bellow order by the decreasing of the absolute error!

word	exact_val	expected	abs_err	rel_err
fixure	1	27	26	26.0%
match	2	28	26	13.0%
let't	1	27	26	26.0%
stir	1	27	26	26.0%
afflict	1	27	26	26.0%
chizzell	1	27	26	26.0%
forbeare	2	28	26	13.0%
ruddinesse	1	27	26	26.0%
wet	1	27	26	26.0%
stayne	1	27	26	26.0%

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

On this run 657 distinct words were counted. The maximum absolute error was, as expected, 26. Ninety of these counters, mostly the ones with higher countes, had 0 absolute error! The number of comparisons was 45931 instead of the exact counters n=26880. It's importante to note that if we increase the k the number of words stored will increase, the number of comparisons decreases substantially, for k=10000 -> comps=32564, and the absolute error decreases to at most 2 and the number of entries in the index with 0 absolute error are 1086, higher then the number of all entries with k=1000. The only noticable way of obtainig a significant execution time diference is by usin k=1, execution time of 0.1s vs k=10000's executions time of 0.01s. This is because the n is very high compared to k=1 so the $delete(ci < \frac{n}{k})$ function is almost every iteration of the loop and producing only one result with a relative error of 13439.0%! To conclude we must choose a relatively high value of k for a large n word text!

III.2 ODISSEY - HOMERO

In this second example, in the English version, it's easy to conclude that with n=133590 if we use the lossy counter with k=1000 the max relative error jumps to 133.59% wich might not be acceptable if we wish to use the less frequent words counters. But as seen in the last example the top results have 0% relative error and with the increase of k, more top results will keep this property.

word	exact_val	expected	abs_err	rel_err
attempting	1	134	133	133.0%
ideas	1	134	133	133.0%
froude	1	134	133	133.0%
science	1	134	133	133.0%
luckily	1	134	133	133.0%
specially	1	134	133	133.0%
doubtful	1	134	133	133.0%
don	1	134	133	133.0%
t	1	134	133	133.0%
tie	1	134	133	133.0%

As expected the absolute error is no larger than $\left\lfloor \frac{n}{k} \right\rfloor = \left\lfloor \frac{133590}{1000} = 133.59 \right\rfloor = 133$! Since the number of words increased by $\frac{133590}{26880} \approx 4{,}97$ times from the last example, using the same k=1000 yields only 78 zero absolute error counters in the 709 distinct words counted. Increasing the value of k to 10000 yields:

1. Max absolute error of 13;
2. 170276 comparisons instead of the 225489 comparisons for k=1000 and n=133590 comparisons of an exact counter;
3. Similar execution times;
4. 590 counters with zero absolute error.

IV. CONCLUSION

This method of counting is specially useful for large input data streams! It provides values that are never underestimated and overstiamted at most by $\left\lfloor \frac{n}{k} \right\rfloor$ which provides us a way of defining a maximum error threshold that will define the paramether k to be used. It's important to note that for high values of n and low values of k will return a small index but with a very high error and that as the value of k is increased the index increases quickly. As expected there are tradeoffs for using this algorithm but it is very useful for big data. The complete tables mentioned are available in the report folder.