Streaming Methods to Evaluate Books' Readability

Siyi Yang A0151298Y Zhendong Liu A0159369L

November 14, 2016

1 Introduction

Before we read a book, we may want to know the readability of the book. Easy reading helps learning and enjoyment**?**, and for educational purpose we also want to give the readability level to reading books so that students can choose the best fit ones to read.

Our goal in this project is to use streaming methods to give a precise enough readability level of a given text book. By streaming method, we achieved less time and space to evaluate the readability of a book. In the streaming method, We designed a score system to give readability score to books so we only update the score as the stream goes. In this project, we test 49 world classic literature books with our algorithms to see the accuracy of our algorithm, and tested another 1254 books to see the time and space cost of our algorithm.

We have compared our algorithm with traditional non-streaming implementation. The result shows our algorithm dramatically reduce the space cost on the large data set. And also spend less time. At the same time, our algorithm still maintains the good estimation accuracy. For detailed information of the improvement please refer to section 6.

1.1 Platform

We run our code on a Mac laptop with the specification below:

- **CPU**: 2.4GHz Intel Core 2 Duo
- **Memory**: 4GB 1067MHz DDR3
- **Operating System**: Mac OS
- **Compiler**: Clang 800.0.42.1 (with -O2 option on)

1.2 Source Code

The source code is on Github:
`https://github.com/lzddzh/cs5234/tree/master/miniProject`

2 Problem Description

Our problem is that given a stream of text S consists of s words w_i, where $i \in [1, s]$ and w_i is a token consist of continues alphabets or w_i is simply a comma or full stop, i.e.

$$w_i \in \{[A \cdots Z, a \cdots z]^*, comma, full\ stop\}.$$

Evaluate the readability R of stream S use as less time and space as possible. The definition of R is given in section 4.

2.1 Readability Of Books

Readability is the ease with which a reader can understand a written text **?** For example, when you read a simple written paragraph, you will easily understand it, but when you face a difficult paragraph, you will do more mental work on thinking the meaning of the texts. If the paragraph is extremely difficult then reader with limited reading ability might totally fail on understand the meanings of the texts.

In our practise, we find the following 3 features of the text has great impact on the readability of a book.

- **Vocabulary**: (a) Huge vocabulary reduce the readability (b) Difficult words reduce the readability
- **Grammar**: (c) Long sentences reduce the readability.

To understand a book or a paragraph, first you have to understand the words in the book. Obviously books written with huge vocabulary is more difficult than books written in a small set of vocabulary. So here we will count the distinct words in a book.

Difficult words will reduce the readability of a book. The difficult book should have higher average word difficulty. On the other hand, a word with more syllables is usually harder than the less one **?**. Thus counting the number of syllables of all the words and further doing some analysis would reflect the difficulty of the words. We use a simple way to estimate the number of syllables and then calculate the variance of such data streamingly as the evaluation of the words' difficulty.

The longer the sentence is, the more difficult for reader to understand it. When we reading we are more likely to focus on continuous words and once we meet a full stop or a comma then we sum up all the read words and try to understand them, so as the sentence become longer, you will have more information to process at the full stop or comma.

Previous researches also shows these criteria has good estimation of the book's readability. For example **???** used the average sentence length and the average length in syllables to measure the readability of text.

3 Data Set

We have two part of data set: One is from a series book which contains 49 books with readability level labelled by publisher; Another is a huge e-book set from project Gutenberg without readability labelled. And for comparing the readability, we also test a REALLY hard book in addition.

3.1 Oxford Bookworms Series Book

The data set we use is the Oxford Bookworms series books**?**, which consists of 49 books written in 6 level of readability. These books are re-written by Oxford ELT to fit different readers with reading ability. Below we list the books with their readability level.

Readability	Books Name
Level 1	Under The Moon, Love or Money, The Coldest Place on Earth, The Monkey's Paw, The Elephant Man, The Phantom of The Opera, The Witches of Pendle, Mary Queen of Scots
Level 2	William Shakespeare, Robinson Crusoe, The Love of A King, Five Children and It, Huckleberry Finn, Alice's Adventures in Wonderland, Anne and Green Gables, Dead Man's Island
Level 3	A Christmas Carol, The Wind in the Willows, The Star Zoo, Tales of Mystery and Imagination, The Secret Garden, The Call of The Wild, Alice's Adventures in Wonderland, Kidnapped, Tooth and Claw, The Bionte Story, Frankenstein, Chemical Secret, The Prisoner of Zinda, The Piciure of Dorian Gray,
Level 4	A Tale of Two Cities, The Hound of The Baskervilles, Gulliver' Travels, The Unquiet Grave, Three Men in A Boat, Little Women, Treasure Island, Dr JEKYLL and Mr Hyde, The Thirty-nine Steps, Silas Marner, Black Beauty
Level 5	Great Expectations, David Copperfield, Far from the Madding Crowd, Wuthering Heights,
Level 6	Oliver Twist, Tess, Jane Eyre, Pride and Prejudice

Table 1: Oxford Bookworms Series Books

These levels are all given by Oxford ELT according to the readability of the books. Books in lower level is easier than books in higher level.

We get all the books in plain text stored in .txt format file from the Internet. After pre-process, we only retain the alphabets and three symbols space, comma and full stop. These book totally have 649777 words, occupying 5.2MB.

3.2 Books from Gutenberg

Gutenberg is a project that offers a great number of free e-books for research use. These books are all real world books taken from free epub books, free kindle books. Since most of the books were previously published by bona fide publishers, and were digitized and proofread by thousands of volunteers, the quality of the books are ensured.

In our project, we downloaded 1254 books from Gutenberg website, which is 519.5MB in total. In later section we will test our algorithm on this huge data set to see the speed and space performance of our algorithm.

3.3 Special Case

Except for above data set, we also tested our algorithm on an special case, in which the book is extremely difficult. The book we choose is *Wealth of Nation*?.

4 Design

To quantify the vague and subjective term readability, we use a score R to represent it. Recall in Section 2.1 we have pointed out 3 important features of texts in book, which are (a) huge vocabulary reduce the readability, (b) difficult words reduce the readability and (c) long sentences reduce the readability. Correspondingly we summarized three reasonable criteria for evaluating the read of a book, as following.

a. **Number of distinct words**: the distinct words amount in the book $D = |\{w_i\}|$

b. **Syllables' variance**: the variance of the syllables of all the words $W = var[syl(w_i)]$ where syl counts the syllables of a word.

c. **Average sentence length**: the ratio of the number of words (with repetition) and the number of full stop symbols $L = \frac{s}{\text{number of full stops}}$.

We combine these 3 criteria into a weighted sum R. Note that these criteria have very different scales, thus we choose such coefficients making all 3 values fit into $[10, 100]$.

$$R = 0.02D + 5W + L.$$

To gain the three criteria D, W and L, we designed three separate parts of our streaming algorithm, which grade on a text stream and reflect the corresponding criteria.

4.1 Distinct words counting

For the first criteria, we only need to count the distinct words in a book. Or equivalently, count the distinct elements in a multi-set as a stream. There are lots of algorithms for such problem, e.g. *Hit Counting*, *Adaptive Sampling*, *Min Count* **?** and *HyperLogLog* **?**. The algorithm we used is the well-known HyperLogLog algorithm, which is an improved minimum-count algorithm.

To be precise, let n be the number of distinct words, namely $n = |\{w_1, w_2, \cdots, w_s\}|$ and let these elements be $e_1, e_2, \cdots, e_n$. The intuition behind minimum-count algorithms is that each sketch carries information about the desired quantity. For example, when every elements e_j is associated with a uniform random variable, e.g. a hash function $h(e_j) \in [0, 1]$, then the expected minimum value of $h(e_1), h(e_2), \cdots, h(e_n)$ is $\frac{1}{n+1}$. The hash function h guarantees that $h(e_j)$ is identical for all the appearances of e_j. Thus, the existence of duplicates does not affect.

And further more, HyperLogLog algorithm is an improvement and offers the better performance. It uses the integer random variables within $[0, 2^p]$ (in our design, p equals 32) instead of real number, which is more computer-friendly. And it counts the largest number of the position of the leftmost 1-bit in binary representation of a number (e.g. $\rho(0001\cdots) = 4$, $\rho(01\cdots) = 2$). But actually it is equivalent to count the minimum number, since the more leading zeros a number has, the smaller it is. The critical defect of above minimum-count algorithm is that the variance is too large, which causes unstability. One idea is that repeat the algorithm via different hash functions and take the mean. And another idea, which we would use in HyperLogLog, is that divide all the numbers into m different groups, calculate the answer for each group, and finally take the harmonic mean.

By analysis **?**, we also need to normalize our result with coefficient

$$\alpha_m = (m \int_0^\infty (\log_2(\frac{2+u}{1+u}))^m du)^{-1}.$$

The theoretic relative error of this method is $\pm\frac{1.04}{\sqrt{m}}$. See the implementation section and result section for more detials.

4.2 Variance method according to syllables

For the second one, basically, we want to do something with the word's syllables. The idea behind is that if most words are just one-syllable or two-syllable words, with high probability we could say that it is an easy-reading book.

So, firstly we need to calculate the number of syllables of a word. Basic rules for syllabication could help us. However as we known, there are many exceptions in the real English world. Such calculation can't be accuracy. Thus we did something much simpler but more effective. We simply count the number of vowels as an estimation of the number of syllables.

Naturally, the average of all these numbers may give us a good evaluation. But in the practice, just take the mean of these numbers is not a good measure. The reason is that, even in the difficult book, most words are just one-syllable or two-syllable words such as “a”, “the” and “of” as the Figure 1 shows. There isn’t a big enough gap between easy book and difficult book.

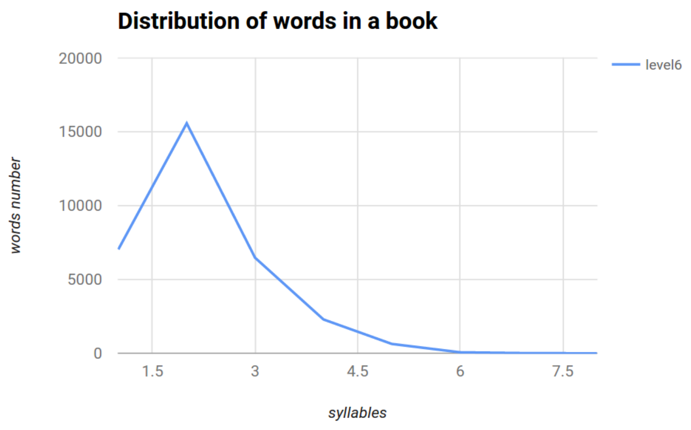

Figure 1: Words in a level 6 book distributing on syllables number

However, the variance does. Different from the mean, just a few multiple-syllable words can make nonnegligible contribution to the variance. Thus we use the variance of the number of syllables as our evaluation of the words’ difficulty.

By the definition, the variance can not be calculated streamingly using limited space. But we can do some tricks as following (let $\sigma = \mathbb{E}[x]$).

$$\begin{aligned} var(x) &= \mathbb{E}[(x-\sigma)^2] \\ &= \mathbb{E}[x^2 - 2\sigma x + \sigma^2] \\ &= \mathbb{E}[x^2] - 2\sigma\mathbb{E}[x] + \mathbb{E}[\sigma^2] \\ &= \mathbb{E}[x^2] - 2\sigma^2 + \sigma^2 \\ &= \mathbb{E}[x^2] - \sigma^2 \end{aligned}$$

The square of the expected value and the expected value of the square can both be maintained streamingly using only $O(1)$ space and linear time.

4.3 Average length of sentences

The last goal is evaluating the difficulty of structure of each sentence. Obviously, it is not an easy task. Thus, we tried some simple ways to solve this problem. The most effective and surprising one is that just take the average length of each sentences.

Although, length is very different from difficulty, but it is a good approximate measure **?**. As we know, complicated sentences always have subordinate clauses and much more decorated words. Note that length here means the number of words, e.g. $len($“We are students.”$) = 3$.

The algorithm is relatively simple. Just maintain two sketches, one for words, another one for full stops. Thus such algorithm runs in linear time and constant space.

5 Implementation

Some more details of our implementation which is not mentioned in above sections are described here. And the brief pseudocode of our implementation are listed below.

5.1 Distinct words counting

Let $h : \mathcal{W} \to \{0,1\}^{32}$ be the hash function hash from words $\mathcal{W}$ to binary 32-bit integer. We implemented our h as following:

a. Choose a random number $b > 26$, and then convert word v into a b-based number, e.g. $h(\text{“}bob\text{”}) = \overline{bob}_{31} = 2 \cdot 31^2 + 15 \cdot 31 + 2$. Note that “a” should be considered as 1 instead of 0. Otherwise there is no difference between $h(\text{“}a\text{”})$ and $h(\text{“}aa\text{”})$.

b. Choose three random numbers k, c, p, return the value of $k \cdot \overline{v}_b + c \bmod p$ as the hash value of $h(v)$.

It costs linear time on the length of the word.

And let $\rho(s)$ be the position of the leftmost 1-bit of s: e.g. $\rho(1\cdots) = 1, \rho(0001\cdots) = 4, \rho(0^K) = K+1$. This function could be simply implemented by enumerating all the binary bits from the highest one to the lowest one, which costs $O(\log m)$ time. The time complexity and space complexity of this sub-algorithm are $O(s \log m)$ and $O(m)$. Following is the pseudocode.

Algorithm 1 Count Distinct Words

```
function HyperLogLog_Sketch($\mathcal{M}$ : a stream of words from $\mathcal{W}$)
    $M[0], \cdots, M[m-1] \leftarrow 0$
    for $v \in \mathcal{M}$ do
        $x \leftarrow h(v)$
        $j \leftarrow 1 + \langle x_1 x_2 \cdots x_b \rangle_2$
        $w \leftarrow x_{b+1} x_{b+2} \cdots$
        $M[j] \leftarrow \max(M[j], \rho(w))$
    end for
    return $E \leftarrow \alpha_m m^2 \cdot (\sum_{j=1}^{m} 2^{-M[j]})^{-1}$
end function
```

5.2 Variance method according to syllables

Let $r : \mathcal{W} \to \mathbb{N}$ be the function which could calculate the number of syllables of words in $\mathcal{W}$. And we need maintain three counters: one for the number of words, one for the linear sum of syllables, and the last one for the square sum of syllables.

But how can we implement such r? Roughly, we estimate such number by counting the number of vowels. Since according to the basic syllabication rule, each syllable should consist of at least one vowel. But we still implement some other exceptional rules, including the tailing voiceless consonant, character “r” following another vowel, and two consecutive vowels. If you are really interested in, please refer to our code implementation.

The time complexity and space complexity of this sub-algorithm are $O(s)$ and $O(1)$. Following is the pseudocode.

Algorithm 2 Variance of syllables

```
function Variance_Sketch($\mathcal{M}$ : a stream of words from $\mathcal{W}$)
    $expect, square, n \leftarrow 0$
    for $v \in \mathcal{M}$ do
        $x \leftarrow r(v)$
        $square \leftarrow square + x^2$
        $expect \leftarrow expect + x$
        $n \leftarrow n + 1$
    end for
    return $E \leftarrow \frac{square}{n} - (\frac{expect}{n})^2$
end function
```

5.3 Average length of sentences

The implementation of this sub-algorithm is quite easy, just maintain the number of words and the number of full stop. The time complexity and space complexity of this sub-algorithm are $O(s)$ and $O(1)$. Following is the pseudocode.

Algorithm 3 Average sentence length

```
function LENGTH_SKETCH($\mathcal{M}$ : a stream of words from $\mathcal{W}$)
    $word, stop \leftarrow 0$
    for $v \in \mathcal{M}$ do
        if $v$ is a full stop then
            $stop \leftarrow stop + 1$
        else
            if $v$ is a word then
                $word \leftarrow word + 1$
            end if
        end if
    end for
    return $E \leftarrow \frac{word}{stop}$
end function
```

6 Results

We have tested our program on two data sets. While one is labelled with authorized readability level **?**, another does not but contains a big number of books. We tested our accuracy performance by using the Oxford Bookworms data set, while the time and space performance is tested by the Gutenberg data set.

6.1 Accuracy Performance

Our algorithm has achieved good estimation of the readability of the books. We applied our algorithm on the Oxford bookworms series books as Figure 2 shows. In the graph, books in y-axis is sorted by their level, so the higher books are more difficult to read. We can see that our R values are almost monotone.

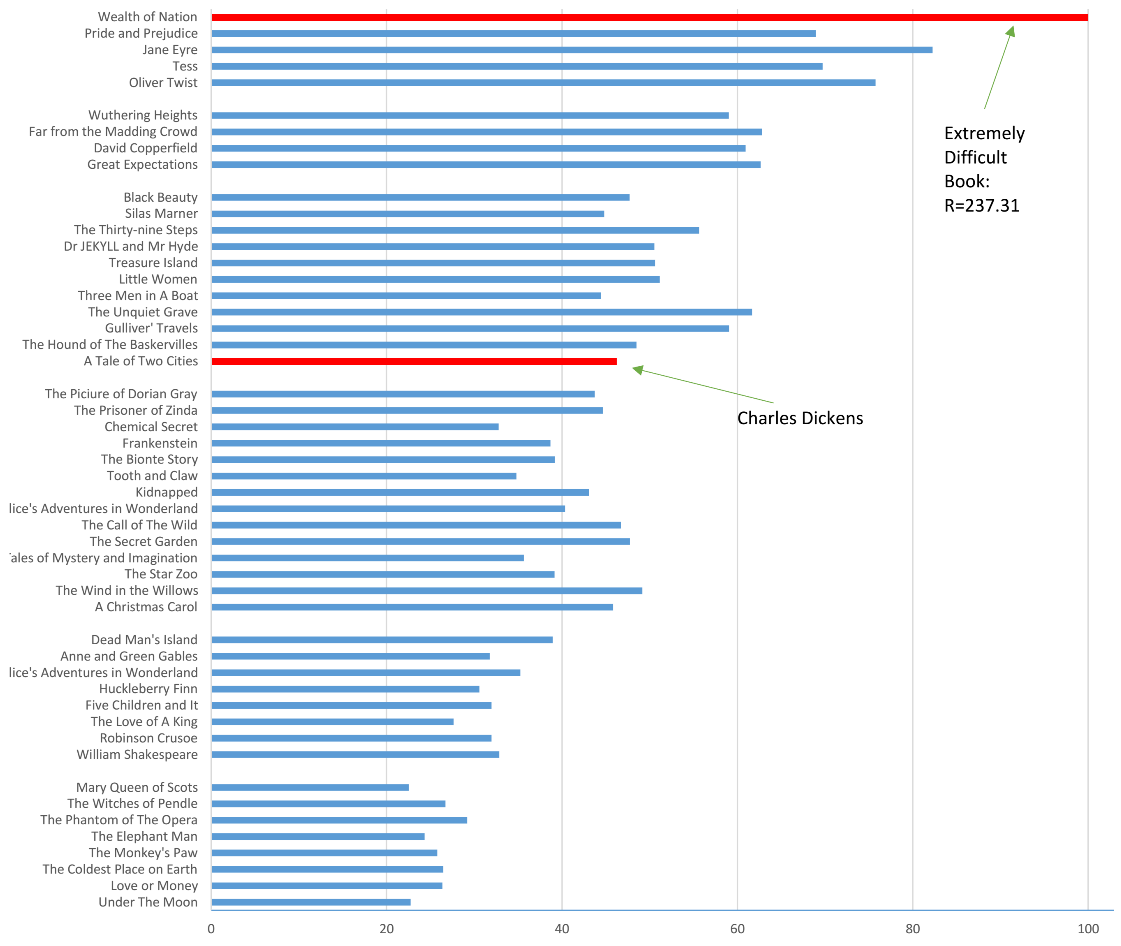

Figure 2: 50 books R values, y-axis sorted by level (m = 16)

Except for Oxford bookworms series book, we have tested other special cases, the book *Wealth of Nation*, which is known to all that this book is very very difficult. We can see is R value in Figure 2, which is much higher than all other books. Another special case is book with long sentence but is not really hard: Charles Dickens's book. We can see his book ranks at the middle of all the books.

6.2 Space Performance

We test the space performance by comparing our streaming implementation and non-streaming implementation. We run both of them on the Gutenberg data set. The non-streaming implementation used a C++ STL container *set* to count the distinct value. To test the space of a running program used is not easy. However, the theoretical space cost is usually consistent with the real cost. Here we know the structure of STL container *set* and the data it processed, so we wrote a script to calculate the space overhead of *set*, the value in the non-streaming row is the result.

Average s = 6894.3	m	Space Overhead	Average Accuracy
Streaming	16	64 bytes	95.67%
Streaming	64	256 bytes	97.63%
Streaming	256	1024 bytes	98.92%
Streaming	1024	4096 bytes	99.40%
Non-Stream	-	387347 bytes	100.0%

Table 2: Space and Accuracy in Hyper Log Log

Table 2 shows the result of streaming and non-streaming methods running on 1254 books. We can see that using the streaming method, only 4096 bytes can reach 99.4% of the full accuracy. Comparing to the non-streaming cost, it is a big improvement. In our practice, using $m = 16$ is accurate enough.

6.3 Time Performance

On the test data set Gutenberg 1254 books, our streaming method reduce the total processing time by 26.07%. The time is mainly saved by Hyper Log Log algorithm.

	Streaming	Non-Streaming
Totall Time	98.91 second	130.71 second
Input Time	5.01 second	4.9 second
Processing Time	93.00 second	125.81 second

Table 3: Time Spend on 1254 books (Data 519.5MB, m = 16)

7 Discussion

Overall, our project succeed in reflecting the readability of a given book by streaming method. Our project shows a great example of applying streaming method to gain more space and time efficiency. And also we should realize this is done by sacrificing certain accuracy.

In real world, not all problem can be solved by streaming. Only those are not sensitive to accuracy and with limited space resources fits the streaming algorithm. There is a very large gap between non-streaming algorithm and streaming one. They focus on very difficult goals. For a specific problem, designing such a streaming algorithm is not that easy. Even if you successfully design one, you still have to carefully treat its trade-off between correctness and memory-saving.

A defect of our project is that although we reduce the space cost greatly, it turns out that it may not be so useful. Because modern PC can easily process an arbitrary book with non-streaming method since a book contains only few data. Streaming algorithm is certainly powerful for text-processing, but restricting to our task, traditional method also works. The reason is that even if you have thousands of book to be processed, you only need to memorize one book. When you finished one book, you could release the memory and then be ready for the next one. Thus even we have a data set with 500 MB size, the traditional method only need about 1 MB memory. At some point of view, traditional method also be made into partly streaming as above discussed. However, our algorithm still shows much much better performance. Thus if you have one book with about 100 MB size, the traditional one will not be so acceptable.

As data volumes grow nowadays, the algorithms we learnt from this class become much more powerful on large data set. Designing an algorithm balance between the space consumption and correctness is really fantastic.