COMP521-14A Assignment 1 Implementing a Simple Decision Tree Learner using Java 8 Lambdas and Streams

YiChen Hu, 1127826

April 14, 2014

1 Introduction

This assignment aims to reimplement Id3 decision tree learner using lambda expressions and streams. A comparison of speed is done once the implementation is proven to generate the same output as the convential implementation of Id3 decision tree.

2 Method

As required in assignment requirement this implementation will: (a) not use any kind of loop or the forEach method available for streams, (b) all lambda expressions must be single-line expressions (i.e. lambda expressions whose right-hand size is en-closed in curly braces are not permitted), and (c) the new code must be free of variables.

A test file generated by BayesNet is used to test if the new implementation generate the same output as the original decision tree.

Then a parallel version of the stream implementation will be created by replacing all stream() method with parallelStream(). This method is not optimal and will be discussed and improved on the next section.

Finally four test files will be used to measure the time consumption of each version. Test files will be generated by BayesNet, LED24 and RDG1 from WEKA [1].

3 Experimental results

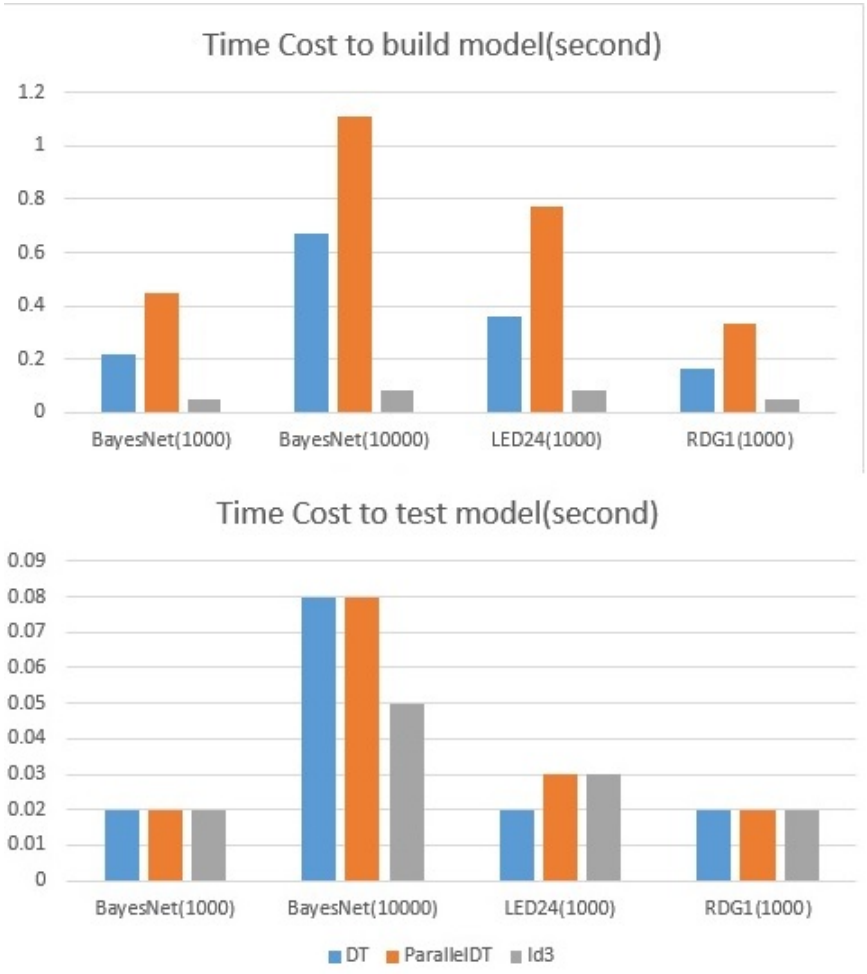

Figure 1: Time costs for 3 versions of ID3 Decision tree

As figure 1 shows, the functional implementation of Id3 decision tree (coloured in blue) is much slower than the original Id3 (coloured in gray). Reasons contribute to this result could be: 1, the lack of temporary variables, which caused some functions to be called redundantly to generate the same variable. Such as the function "getAttributeWithMaxGain". 2, The stream versions come with a lot of overheads such as converting collections to streams. For example, to compute the class value of a set of instances, the function has to convert collection to stream twice, and implement a comparator object, which is unnecessary in

an ordinary loop design. 3, the implementation of stream versions were hastily done by a novice learner of java 8 lambda expressions, and developed with constraints(as listed in above section).

Also, the figure 1 indicates that if one simply replace stream() by parallelStream(), the time cost can be largely increased instead of decreasing. The parallelStream() creates overheads to prepare the stream to be parallel capable such that if the dataset is often small, the parallel stream will be a drawback.

The time costs for testing model are quite similar and because the base scale is 0.01 second, the absolute difference is not very significant.

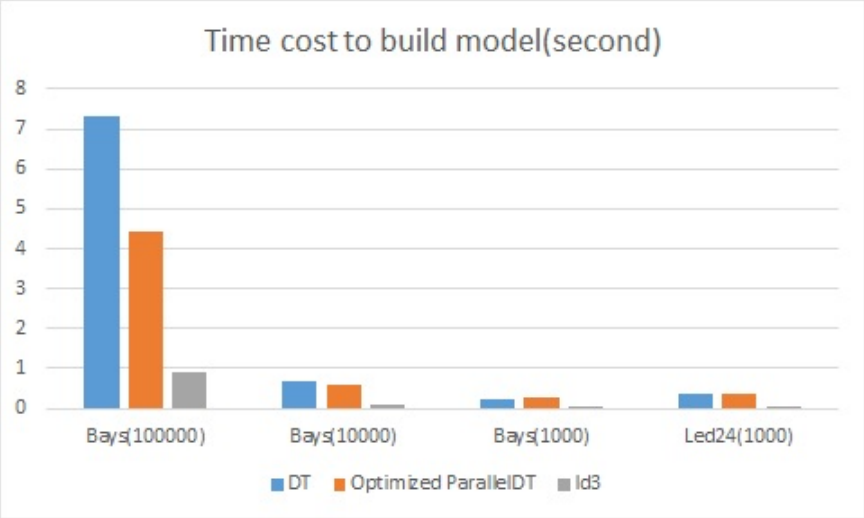

Figure 2: After optimization of parallel stream implementation

Figure 2 shows an improved parallel stream implementation. In this case the parallel stream version has similar performance to the ordinary stream build(DT) when data set is small, and significantly over-performs DT when data set becomes larger. However, the parallel stream version is still a lot slower than the ordinary Id3.

4 Conclusions

Pure functional programming may not be a good idea.It is either slow in processing and poor in code readability.
Stream parallelism can improve the processing speed however it has to be used at the right time, otherwise it will be a drawback.
The potential of parallel stream has not been fully tested out in this assignment

due to the constraints such as "no-variables", need more experiments to see if and how much the parallel stream could outperform the conventional loop design.

References

[1] Witten, I. H., and Frank, E. Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann. 2005,