Big Data Analytics Programming

Assignment 2 – Efficent Decision Tree Learning

Ivan Lovrenčić - 0827467

Implementation:

The final implementation of the CART algorithm consists of the following optimizations. First, I implemented the parallelization of the construction of the decision tree. I used the std::async function to asynchronously and recursively build a tree. This implementational trick saved a lot of time, as I could use all of the CPU potentials to create a tree. Secondly, I implemented the suggested presorting of the dataset. For each feature, the dataset is sorted based on the values from that feature. The sorting was performed for both numerical and categorical features. Lastly, I perform a linear scan of the dataset and dynamically update the count matrices. With these optimizations, the average build time for one tree is 200 seconds or 3.3 mins. In figure 1., we can notice how the computation time decreased as we added specific optimizations.

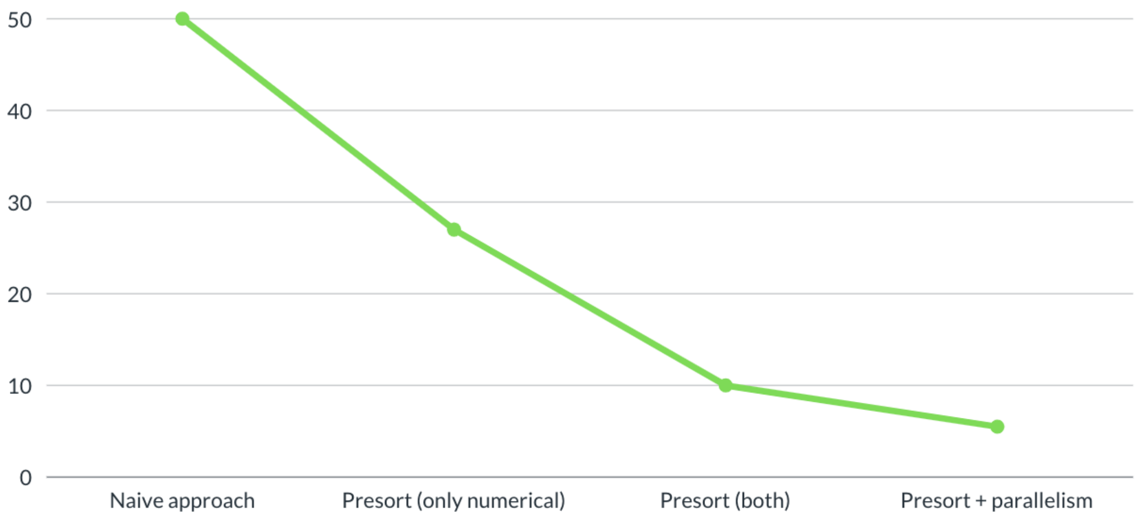

Figure 1. Computation time (all values are in minutes)

Bugs:

The main bug/difficulty was with the memory management during multithreaded calls. Specifically, in asynchronous calls, although we pass object references, the compiler will create copies of the objects, which will result in considerable memory overhead. To fix that, I wrapped all function objects with std::cref(), which, instead of references, passes a pointer to that object. This trick vastly minimized my memory usage. Another minor bug/slowdown was dataset sorting. In the custom Comparator object, I used the try/catch block to check whether the value is numeric or categorical. However, while optimizing code, I learned that using try/catch block is computationally expensive. I replaced that by adding the extra flag in the Data Reader, and I saved 150 seconds of runtime. Lastly, the most significant memory leak was with the DataReader class. Because the DataReader was "injected" into the Bagging object and DecisionTree object, the memory overhead was huge. I solved this by replacing the object variable with the pointer in those classes. With that implementation trick, I managed to drastically reduce memory usage.

Bagging:

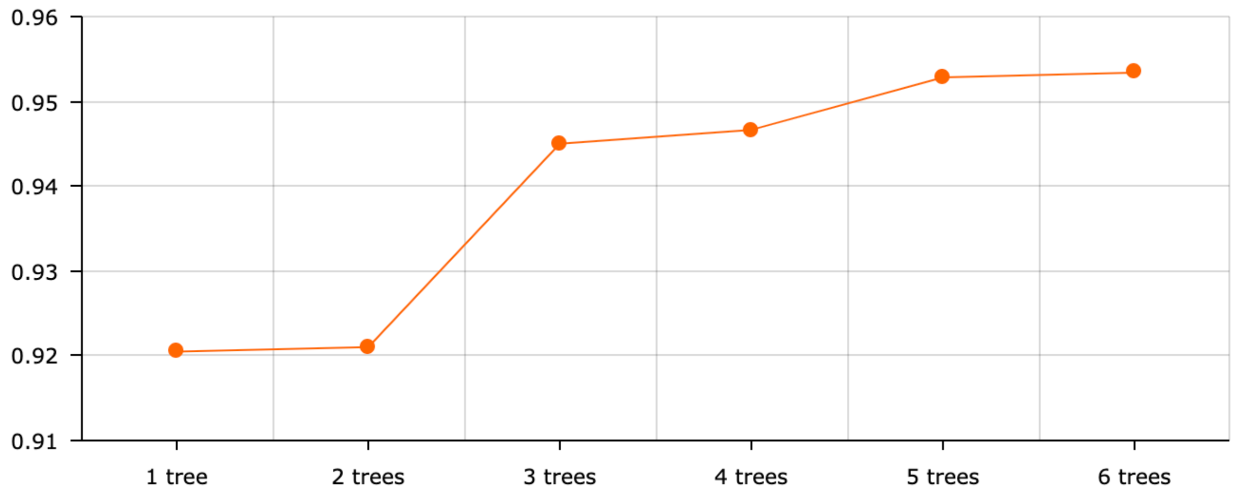

Figure 2. Accuracy with different number of trees in the ensamble (learning curve)

As we can notice for me the illustration above, increasing the number of trees improves the overall accuracy of the models. This conclusion is in line with the hypothesis that bagging makes decision tree models more robust and accurate. As we increase the number of trees, the accuracy will continually improve, but at one point, it will hit a plateau.