ParBag: Parallel Bootstrap Aggregation on Random Forests

Joshua Hahn jyh2134

November 2022

1 Project Overview

This project will implement a parallel version of the Bagging (bootstrap aggregation) learning ensemble algorithm in Haskell. The goal of the project will be to create a classifier that will train and predict on the UCI Iris dataset. The project is comprised of four tasks:

1. Creating a model that can successfully predict labels serially
2. Parallelizing the task and performing the same task faster
3. Evaluating the speedup of the parallelizations using a variable number of cores.
4. (With time) perform pruning to further accelerate program runtime

2 Background

The bagging method is a specific example of ensemble methods: meta-algorithms that learn features by aggregating the predictive abilities of many independent "weak learners" that each have an accuracy only slightly better than random. Bagging is one strategy in ensemble methods that generates independent weak learners from a single dataset by grouping the original training points into distributionally independent training sets.[1] The algorithm can be divided into 3 parts:

1. The original dataset is grouped into n samples, with each bootstrap sample B_b randomly selecting (with replacement) l values from the set.
2. A classifier f_b is trained on each B_b, with accuracy greater than randomly choosing a category.
3. Finally, the model predictor F predicts a class via majority voting (averaging the results of all of the independent weak learners).

For this project, each independent learner will be a decision tree, resulting in an aggregate random forest that will predict the class of the testing data. This algorithm is shown to work under two conditions:

1. The weak learners all learn from (pseudo)-independent distributions
2. Each weak learner has a prediction accuracy greater than statistical randomness

3 Parallelization

Parallelization can be achieved in the first two steps of the algorithm:

1. In the first step, the bagging process can be parallelized by randomly sampling from the original dataset independently. This is possible because the sampling process is done with replacement, meaning each individual bootstrap sample randomly chooses from the same original set, and the selection of one sample does not affect the selection of another sample. However, since this process is such a small portion of the program, I do not believe that this will reduce the runtime of the algorithm by a large factor.

2. The second step, which takes up a bulk of the algorithm's runtime, can also be parallelized. Since each weak learner is trained on a unique bootstrap sample, each of these weak learners can be trained on separate cores without affecting the training of other models.

The third step, in which we combine the "votes" of each weak learner, can also be paralellized by sharding the results of the learners, then performing a divide-and-conquer style aggregation. It is my prediction, however, that this process will only create more overhead and lead to an overall decrease in performance, since the sequential implementation is not a large aprt of the program's runtime anyways. If time remains, I will also conduct a performance test to see if parallelizing the aggregation process results in a significant speedup.

References

[1] Leo Breiman, *Bagging Predictors*. Machine Learning, Volume 24, 1996.