# Multithreading efficiency analysis

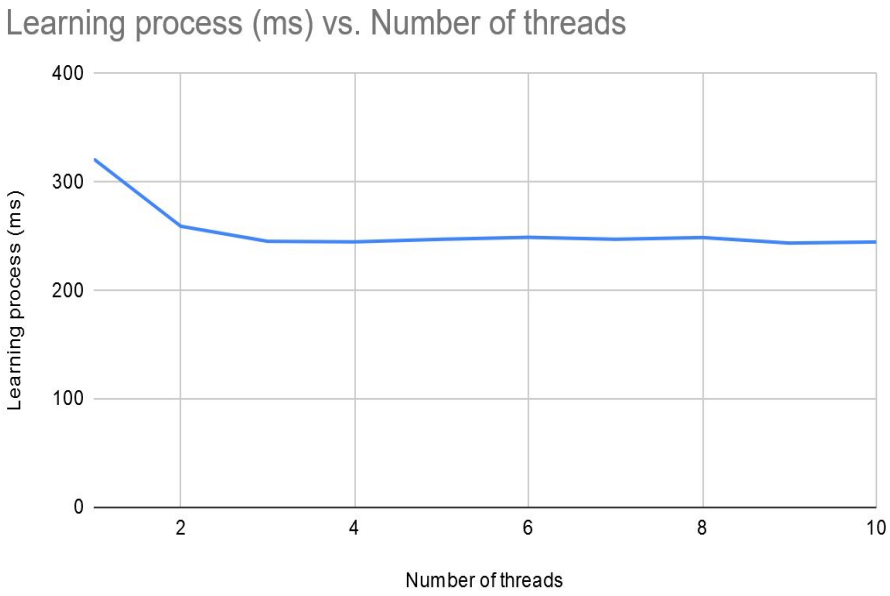
---

Jakub Belter, Data Engineering 07/04/2020

The task was to use multiple threads to increase the efficiency of our program as well as see how the multithreading works in general. The process that I implemented was rather simple. In general we would teach **n** models which reside on **n** threads so that each of them is being taught separately with separate data - the whole list is divided into equal **n** parts.. Then we would calculate an error of the prediction and we would see how many instances were over the board and we were not even close to predicting it. Hence we would see that the offers are not based only on surface but also on the location etc.

In the loop we would create i-thread pool, which would increment up to 10. Inside this loop we would try 10 times how fast our model was trained (so that the values are optimal and not changed because of some weird system action). Then after the training is done we would print the average time it took as well as how many outliers there were. It is printed so that we see that the value is constant (if it was changing then it would mean that there is a synchronization problem).

Here is a graph of time it took to execute the training (averaged over 10 runs):




As we can see there is no much difference after 3 threads which is likely due to the fact that the operation takes too little time to be efficient - setting up threads takes up more time. Also the problem was that I couldn't add more data to the calculation since Java Heap error occurred. Worth noting is fact that the time it took to check error (n-model mean prediction) took linear more time as the number of threads increased which overall caused the overall time to be the same whether using the threads or not.