ISOLATION FOREST USING SPARK

KIND, MATIAS MCARRAS2@ILLINOIS.EDU
AND
HARIRI, SAHAND HARIRIA2@ILLINOIS.EDU

We consider parallelizing the isolation forest algorithm using Spark. The following are six different ways in which we can parallelize the algorithm.

The experiment is performed for a training dataset consisting of 4000 data points distributed as a two dimensional normal distribution with mean $\begin{bmatrix}10\\1\end{bmatrix}$ and covariance $\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$. The same data is used for computing anomaly scores for testing purposed. Figure 1 shows this dataset as well as the distribution of anomaly scores computed using the algorithm. We have added some anomalous points manually.

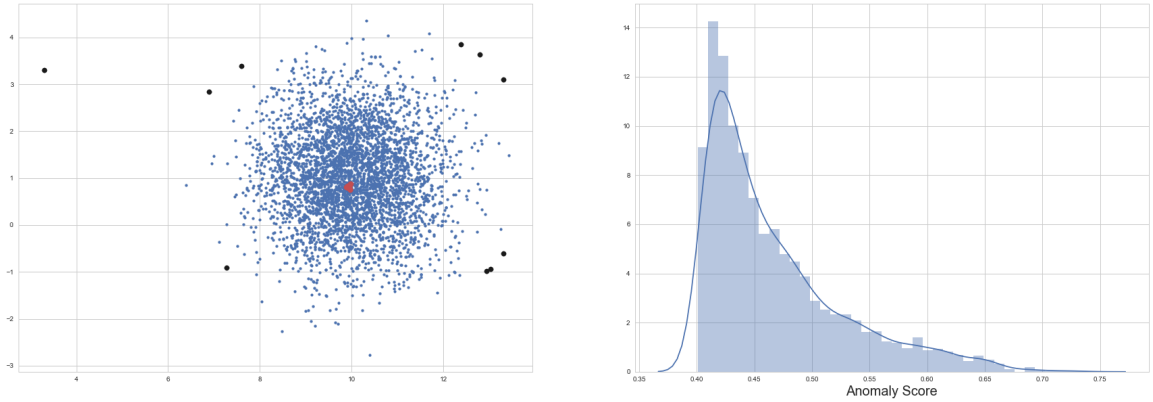

FIGURE 1. On the left, we have the dataset. Some sample anomalous data points discovered using the algorithm are highlighted in black. We also highlight some non-anomalous points in red. On the right, we have the distribution of anomaly scores obtained by the algorithm.

Each case of testing is performed five times, and the mean and the variance of the time taken to perform each trial are reported in seconds in table 1.

(1) **Case 1**: Create 800 trees in a single core and run everything serially.
(2) **Case 2**: Create 100 trees in 8 cores. Each core has access to the full data set for training purposes. For scoring, each core receives the whole data and runs it through forests of size 100. At the end the scores from each forest are aggregated.
(3) **Case 3**: Split the data for training among 8 cores. Each core has 100 trees. Scoring is done similarly to Case 2.
(4) **Case 4**: Split the data for training among 8 cores. Each core has 800 trees. Scoring is done similarly to the previous two cases. This naturally is expected to take the longest.
(5) **Case 5**: 800 trees in a single core. After training, distribute the trees over the cores, and compute the scores in a vectorized way.
(6) **Case 6**: Split the data for training among 8 cores. Score each partition in one of the cores and simply collect data.

TABLE 1. Summary of results.

	Forest Assembly		Computing Score	
	Mean	Var	Mean	Var
Case 1	3.2501	0.0219	71.4731	0.0559
Case 2	0.0063	1.274e-7	16.8996	0.8627
Case 3	0.0073	1.702e-6	21.1470	0.8066
Case 4	0.0064	2.886e-6	163.54	16.169
Case 5	6.6305	0.6377	44.616	6.8809
Case 6	0.0071	8.574e-7	3.2851	0.02851

The data in the table above is summarized graphically in figure 2. Note that the y-axes are not on the same scale.

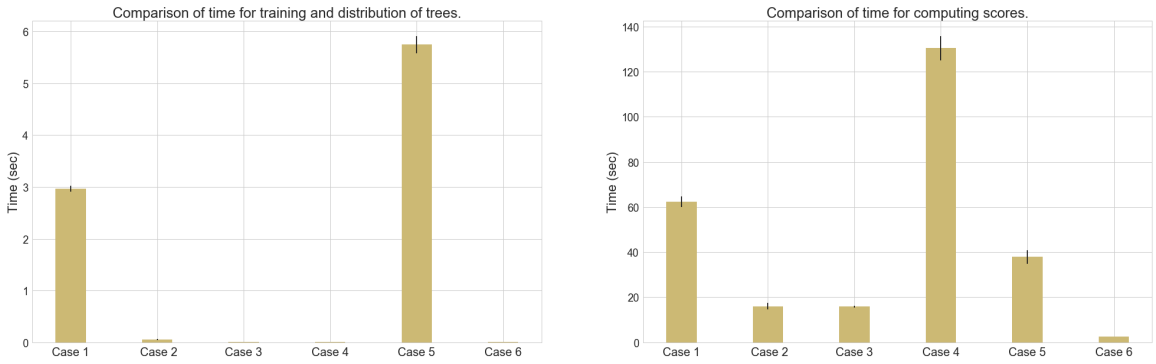

FIGURE 2. Comparison of time taken for training and obtaining anomaly score for each case listed above. The plot on the left shows the time taken to train and distribute the trees among the cores. The plot on the right shows the times taken to compute the anomaly scores using each method.

As another metric, we consider the case of running serially to be the benchmark case. We then compare the anomaly score, S, for each data point obtained using each of our parallelization methods with the benchmark. The measure of the comparison is the following:

$$E = \frac{100}{N}\left(\sum_{i=1}^{N}\left(\frac{S_i^{\text{parallel}} - S_i^{\text{benchmark}}}{S_i^{\text{benchmark}}}\right)^2\right)^{\frac{1}{2}} \tag{1}$$

where N is the number of data points. Figure 3 shows the value of E for each case. Putting these results together with the total time taken for each case (training and scoring) shown in figure 4, we can see that case 3 perhaps offers the best performance.

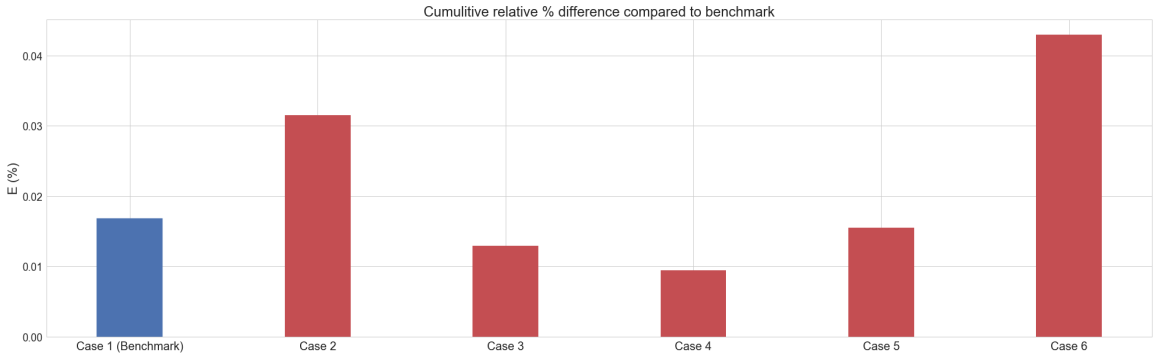

FIGURE 3. Difference computed as shown in equation 1

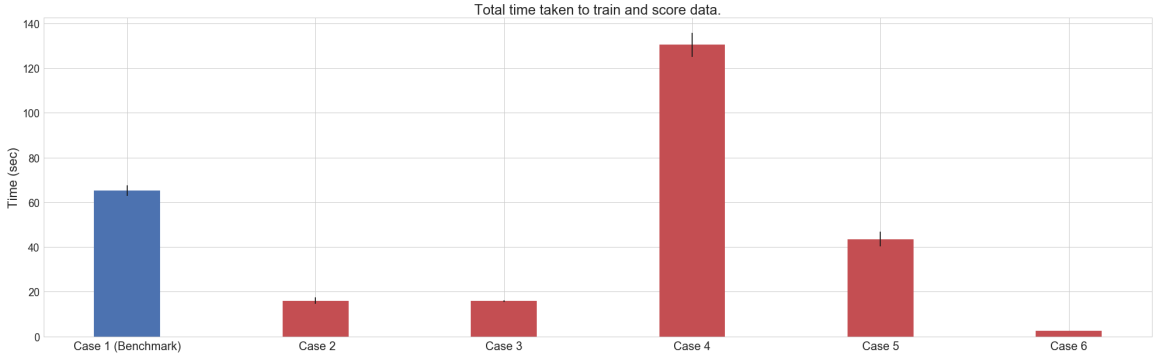

FIGURE 4. Total average time taken to run each case. This includes training, distributing trees, and computing scores.

Another interesting thing to look at is the question of how the difference between each case and benchmark changes as the anomaly score grows. Intuitively the larger anomaly scores should also show the largest discrepancy. In order to take a look at this, we sort the data points by anomaly scores associated with them using the benchmark case, and plot the anomaly score for each in figure 5.

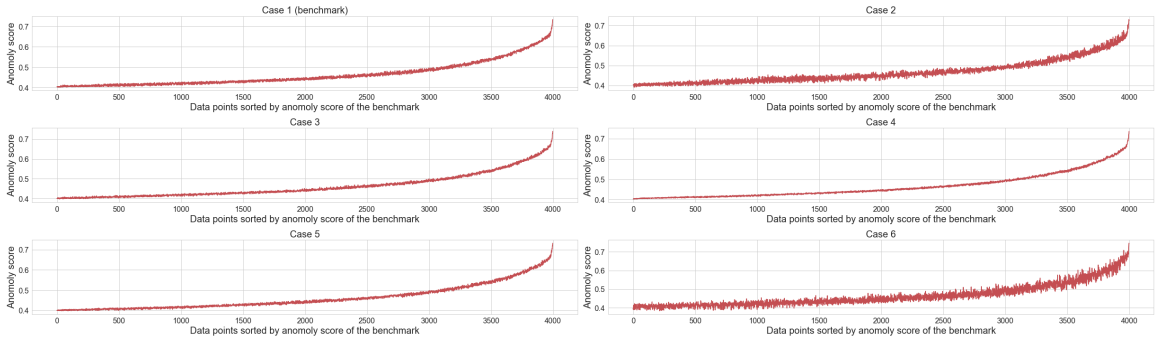

FIGURE 5. We sort the data points in the benchmark case according to their anomaly scores. We then sort the data points in other cases according to the position each one assumed in the benchmark case.

Now taking each case of this sorted list and plotting the relative difference of each case with the benchmark, we have the results shown in figure 6. We can kind of see that as the anomaly score grows, the difference on average also increases in each case (case 2 looks a little shady though).

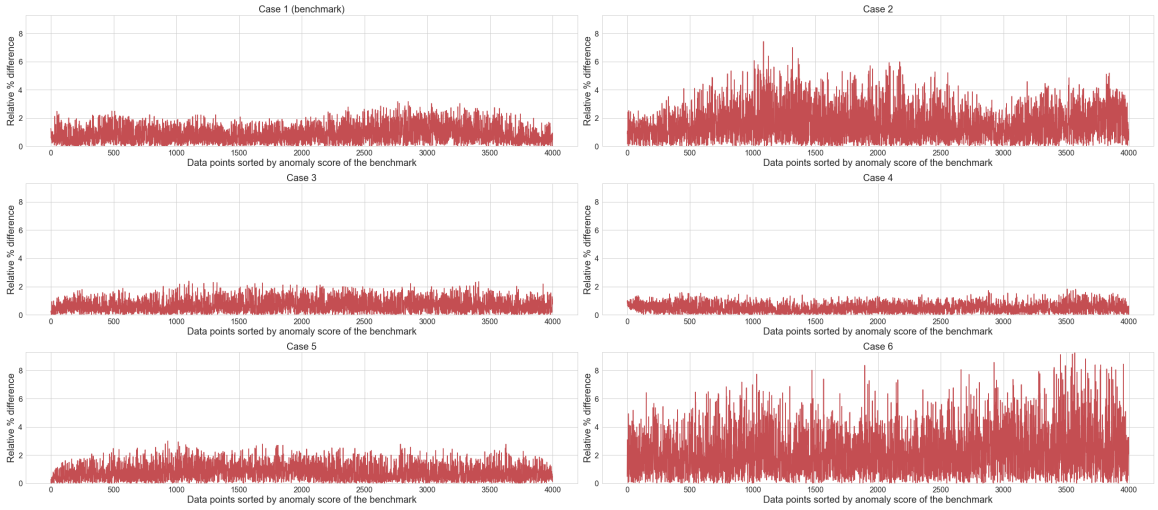

FIGURE 6. We now look at the square difference between data points from each case and the benchmark after sorting them.

Next steps:

(1) Get Spark up and running on a Kubernetes cluster.
(2) Run all of this on the said cluster.
(3) Think about how to train data continuously as new data comes in.

More to come...