Batch and online anomaly detection for scientific applications in a Kubernetes environment

Sahand Hariri*
University of Illinois at Urbana-Champaign
sahandha@gmail.com

Matias Carrasco Kind†
National Center for Supercomputing Applications
mcarras2@illinois.edu

ABSTRACT

We present a cloud based anomaly detection service framework that uses a containerized Spark cluster and ancillary user interfaces all managed by Kubernetes. The stack of technology put together allows for fast, reliable, resilient and easily scalable service for either batch or streaming data. At the heart of the service, we utilize an improved version of the algorithm Isolation Forest called Extended Isolation Forest for robust and efficient anomaly detection. We showcase the design and a normal workflow of our infrastructure which is ready to deploy on any Kubernetes cluster without extra technical knowledge. With exposed APIs and simple graphical interfaces, users can load any data and detect anomalies on the loaded set or on newly presented data points using a batch or a streaming mode. With the latter, users can subscribe and get notifications on the desired output. Our aim is to develop and apply these techniques to use with scientific data. In particular we are interested in finding anomalous objects within the overwhelming set of images and catalogs produced by current and future astronomical surveys, but that can be easily adopted to other fields.

KEYWORDS

Anomaly Detection, Isolation Forest, Cloud Computing, Kubernetes, Apache Spark

ACM Reference Format:
Sahand Hariri and Matias Carrasco Kind. 2018. Batch and online anomaly detection for scientific applications in a Kubernetes environment. In *ScienceCloud'18: 9th Workshop on Scientific Cloud Computing, June 11, 2018, Tempe, AZ, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3217880.3217883

1 INTRODUCTION

The problem of anomaly detection has wide range of applications in various fields including scientific applications. anomalous data can have as much scientific value as normal data or in some cases even more. It has long been a topic of interest and has only become more relevant with popularization of data driven methods in both science and technology. A comprehensive survey of the research dedicated to anomaly detection can be found in [4]. There are many areas of application such as fault detection, failure detection in medicine [15], intrusion detection, exotic sources in astronomy, among others, which can easily be categorized as anomaly detection problems. While various techniques exist for approaching anomaly detection, Isolation Forest [9, 10] is one with unique capabilities. This algorithm can readily work on high dimensional data, it is model free, and it is computationally scalable. New advancements in cluster and cloud computing present a very good platform for developing a comprehensive, multi-purpose, reliable, scalable, and efficient anomaly detection infrastructure using mainly Isolation Forest, either as stand-alone service, or in combination with other techniques.

This service can be used in many areas such as general science, Internet security, and forecasting to name a few. As an example, this will particularly be important for astronomy where finding anomalies has become very challenging because of the characteristic of the data as well as its size [8, 11], and it is usually not well addressed by standard techniques. Future astronomical surveys, such as LSST on the ground[1] or GAIA in space,[2] will generate such amounts of data that basic tasks such as finding anomalies within the data, either from the source catalogs or from the images, can be challenging. At the same time, it can lead to important new scientific discoveries as well as systematic understanding of the effects within the data in order to characterize them. This same example is valid in other scientific fields where vast amount of data is generated and processed and the need for anomaly detection from static or streaming data is unavoidable.

In this paper, we present a streamlined cloud based anomaly detection service framework that at its core uses an extension of the Isolation Forest method, named Extended Isolation Forest, as well as ancillary dimensional reduction solutions to be applied to batch and streaming data. These extensions are used to improve the performance of Isolation Forest as a standalone anomaly detection algorithm. This algorithm will be put to use by robust, reliable, resilient, and scalable cloud service using a Spark [14] cluster managed by Kubernetes [2] [3] to allow a user friendly interaction with data with powerful anomaly detection capabilities. Within Kubernetes, this services can be easily scalable and fault tolerant. It can also be extended to use not only with Spark, but with other big data frameworks as well.As such, the specific goals that we have in mind can be divided into two categories of (1) anomaly detection for scientific applications, and (2) technology development for such infrastructure.

*Corresponding author
†Corresponding author

ScienceCloud'18, June 11, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5863-7/18/06…$15.00
https://doi.org/10.1145/3217880.3217883

[1] https://www.lsst.org/
[2] http://sci.esa.int/gaia/
[3] https://kubernetes.io/

In section 2 we present our advances in anomaly detection algorithm. In section 3, we introduce our framework architecture and the implementation of our infrastructure. Section 4 shows the current status and results, while section 5 finalizes with concluding remarks and future work.

2 ANOMALY DETECTION

We built on Isolation Forest and utilized some of the modern techniques to improve upon current state-of-art anomaly detection. The standard Isolation Forest can readily be used with high dimensional data. It is also easily scalable and fits quite well into the paradigm of MapReduce [5] operations employed by Spark to leverage the advantages that parallelization framework has to offer. However, considerable improvements in the results can be achieved with an extension to the algorithm itself, introduced in [7]. Additionally we can combine this algorithm with different clustering techniques such as self organized maps in order to tackle more complex problems and also to handle streaming data as shown in [6]. In the following subsections, we present a brief explanation of how Isolation Forest and its extension work.

2.1 Isolation Forest

In the standard Isolation Forest, the procedure is similar to a Random Forest algorithm [3] where the idea is to randomly divide subsamples of data into two sections that fall below or above a certain value for a randomly selected feature within the dimensions of the problem. This division takes place recursively on smaller and smaller subsets of the data until single data points are isolated, or a certain preset depth limit is reached. This structure can be thought of as a binary tree. By combining many of these different trees, each produced from a different subsample of data, we form a "forest". Since anomalies by definition are few and different from the rest, they will be isolated, on average, much more quickly compared to the nominal points.

To illustrate this, consider a two dimensional random set of points shown in Figure 1.

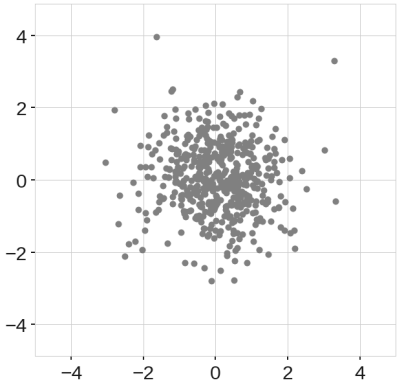
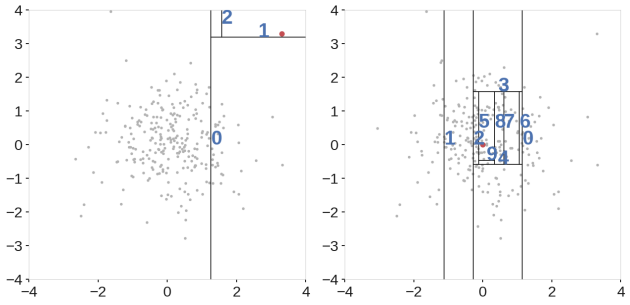


Figure 1: Single normally distributed 2D blob.

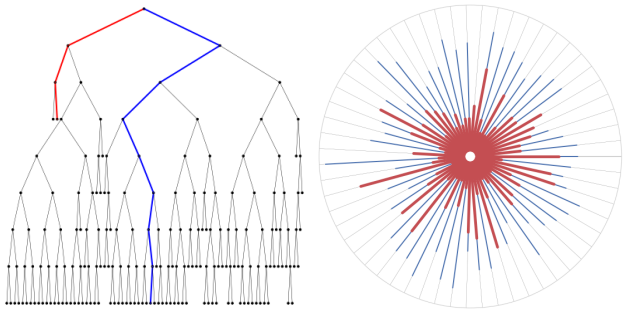
First a subsample of this data is chosen at random. Then we start by choosing one of the random features x or y. Once a random feature is selected, we choose a random value for this feature and compare all the data points from our subsample with this value. Points below the value go down the left branch, and points above go down the right branch. This gives us the first level of branching and partitioning. We then repeat the same process of choosing a random feature and a random value for all the data points in each branch to move further down the tree. In the two dimensional example, these random values of random features translate to straight lines in either horizontal or vertical directions. As we continue this process, we build a binary tree whose branches have depths corresponding to the number of operations required to isolate each data point, unless a preset limit is reached. This process of isolating points is shown in figure 2.



(a) Isolation of an anomaly point. **(b) Isolation of a nominal point.**

Figure 2: Partitioning of the data during training of an Isolation Forest. a) shows the process for an anomaly while b) shows that of a nominal point. There are many more branching processes required to isolate a point buried deep within the center of the distribution.

Anomalous points are isolated quickly whereas nominal points reach far deeper into the tree as seen in Figure 3. An example of a binary tree created as a result of this process is shown in Figure 3a. The red line corresponds to the path taken by the anomalous point from Figure 2a, and the blue line is that of the nominal point in Figure 2b. After building one such tree, we choose another subsample of the data and repeat the same process to build yet another tree. Once many trees have been built, the "training" is complete and we can use this "forest" to score each data point. To obtain an anomaly score, we run a given data point through each one of the trees, and compute the depth of the path traversed by the point in each tree.



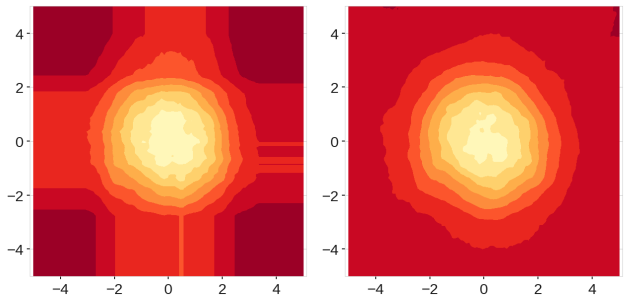
(a) Example Tree **(b) Example Forest plotted radially**

Figure 3: a) Shows an example tree formed from the example data while b) shows the forest generated where each tree is represented by a radial line from the center to the outer circle. Anomalous points (shown in red) are isolated very quickly, which means they reach shallower depths than nominal points (shown in blue).

The aggregate result of this computation from all the trees is then translated into an anomaly score as described in [9]. Figure 3b shows a full forest. Each radial line corresponds to a full depth tree. The red lines are the depths the same anomalous point from figure 3a reached in each tree, and the blue lines are the depths the same nominal point reached in each tree. This visualization illustrates the fact that on average, anomalous points reach far shorter depths compared to nominal points. It is on this basis that Isolation Forest can detect anomalies so efficiently.

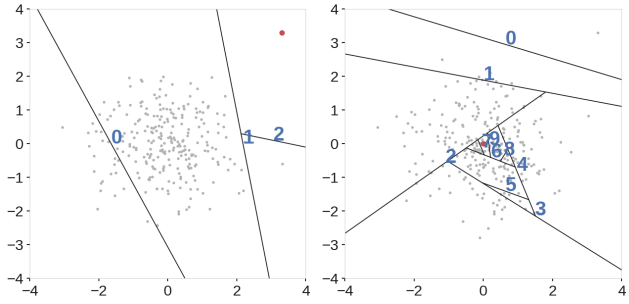
2.2 Extended Isolation Forest

The need for an extension of the Isolation Forest arises from studying score maps for various examples. Here we simply consider the two dimensional example of figure 3. To see a more comprehensive explanation of why this extension is needed and how it works, see [7]. Here we present a brief description. Figure 4a shows the score map for our example obtained using the standard Isolation Forest. For the given dataset we expect to see a score map that has relatively low anomaly scores towards the center, and a gradual increase of the score as we move outward in any radial direction.



(a) Standard IF (b) Extended IF

Figure 4: The improvement of the score map is shown. The artifact of lower anomaly scores bands along x and y axes seen in a) are completely gone in b) where Extended Isolation Forest was used.



(a) Isolation of an anomaly point (b) Isolation of a nominal point

Figure 5: Partition of data during training of an Extended Isolation Forest. The restriction that branch cuts have to be either horizontal or vertical has been removed.

However, as seen in the Figure, the change in the score map depends on the direction we choose to move away from the center which can lead to errors in the classification. On the contrary, we can see from Figure 4b that the score map reflects anomaly scores precisely as they should be. This improvement is achieved using the Extended Isolation Forest (EIF). The idea of EIF is simple. During training portion of the algorithm, we modify the partition process. As shown in Figures 2a and 2b, the "branch cuts" are horizontal or vertical lines. The EIF gets rid of this restriction and allows for these lines to have random slopes with random intercepts as shown in Figures 5a and 5b. The result is a drastic improvement of the anomaly scores no matter which data point is being scored as seen in Figure 4b.

This extension is easily adapted for higher dimensional data and it shows an overall improvement in the performance compared with the standard algorithm. For a N dimensional dataset, the branch cuts can be thought of as $N-1$ dimensional hyperplanes. In the case of the EIF, we choose these hyperplanes by picking a random normal vector and a random intercept point within the data. The various advantages of Isolation Forest in terms of being scalable and appropriate for MapReduce operations remain the unchanged.

3 FRAMEWORK ARCHITECTURE

As we work towards our research goals, we are leveraging state-of-the-art scientific computing technologies while developing new techniques which will allow us to provide a robust, resilient, modern and fast service. In order to streamline the process, we are working on the following:

- Build a simple user interface through which jobs can be submitted and executed in a cloud environment managed by Kubernetes.
- Build a system for automatic deployment for various kinds of jobs, exposed through well documented API endpoints, for batch and streamlined data.
- Build capability for accepting streaming data for real time detection of anomalies during scientific experiments, surveys, etc.
- Develop an alert system which will send notifications to subscribed users in the event of a high anomaly score for a data point.
- Build a visualization framework for display and interpretation of the results of the process.

The very nature of the algorithm lends itself well to parallelization using a MapReduce [5] schema. Recent improvements in distributed computing on cloud clusters present opportunity for significant improvement in anomaly detection using Extended Isolation Forest. A framework that is highly effective in implementing MapReduce is Spark [14], which makes use of a data structure called Resilient Distributed Data (RDD) to distribute data objects across many computing units. We have already implemented a parallelized version of the Extended Isolation Forest using Spark which runs within Kubernetes, and we are looking to augment such approach with complementary interfaces using the same framework.

The cloud platform we used to develop our initial test environment is currently OpenStack on which we first deploy a Kubernetes cluster. The basic idea is to provision a number of virtual machines on OpenStack[4] [12] and run a Spark cluster across these VMs alongside other micro services deployed using Kubernetes. However, there are

[4]https://www.openstack.org

a number of considerations that are noteworthy. We would like to provide a service that is (1) fast to deploy, (2) resilient to node failures, and (3) can be easily scaled up or down. (4) provide similar features for ancillary user interfaces and API services.

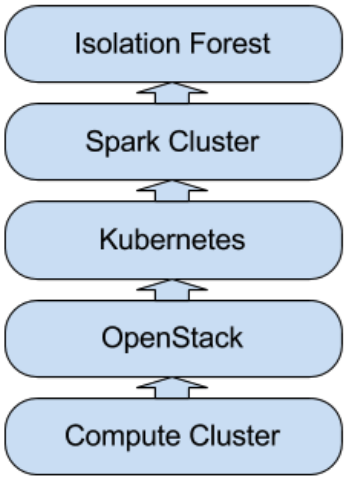


Figure 6: Current Technology Stack.

For the easy deployment, we turn to containerizing our applications and web services. Docker[5] containers provide an excellent solution for packaging up a lightweight minimal working environment that is built to perform a specific task. Building a Spark cluster with Docker containers however, would mean that we need to manage the orchestration of many Docker containers. It also opens up the question of being able to handle node failures as well as the scalability of the cluster. Kubernetes [2] provides a solution for exactly this purpose with additional convenient features. Kubernetes is a container orchestration system that works with Docker that is tasked with managing the networking between different compute units, ensuring containers are re-spawned and joined to the cluster in the event of a failure, deploying updates, scaling the size of a cluster, and many other cloud native utilities. We therefore deploy a Spark cluster which is managed by Kubernetes, with few dedicated pods for master and several others for worker nodes. Aside from this cluster, we also deploy a Jupyter Notebook [1] server using Kubernetes which acts as our main user interface to interact with and submit jobs to the Spark cluster.

It is worthwhile to mention that since Kubernetes can be run on bare metal or other cloud providers like AWS or GKE, there is no real need to use OpenStack. However, because our current institutional infrastructure for computing provisioning uses OpenStack, we have chosen to deploy our Kubernetes cluster on top of OpenStack. In the future, we will potentially move away from OpenStack, and deploy Kuberenetes directly on hardware. Since Kubernetes offers extremely efficient and easy to use capabilities that we desire in building our service, such as resilience and scalability, and since it is rapidly becoming one of the most popular container management and orchestration systems in the world, we believe it is well advised to design our service with Kubernetes acting as the core component regardless of where it is deployed. We have not tested, however, whether the extra layer of abstraction affects the performance in any way, for example by introducing extra latency. We believe this is negligible but worth exploring in the future, especially for very efficient anomaly detection on streaming data.

Figure 6 shows the technology stack and hierarchy in use in our exploratory environment, while Figure 7 shows an overview of the architecture design of this framework. In the User Interface section we can see a Streaming interface, which receives the data handled by a broker interface and runs it through the forest. Users can then subscribe and receive alerts when a new anomaly is detected. Additionally, there is Jupyter Lab and a simple web application User Interface, which also talks directly with the Spark Master to examine data points for anomaly from the user's input. This can also be done through a queue manager to process data in an orderly and asynchronous manner. In the computing stage, we see the Spark nodes with the trees loaded ready to score new data, a Spark Master that orchestrates the MapReduce, and Redis[6] which acts as broker for the streaming service, or for the queue manager. Finally we have a Storage layer which can serve as backup and/or general storage of the training data which can evolve with time.

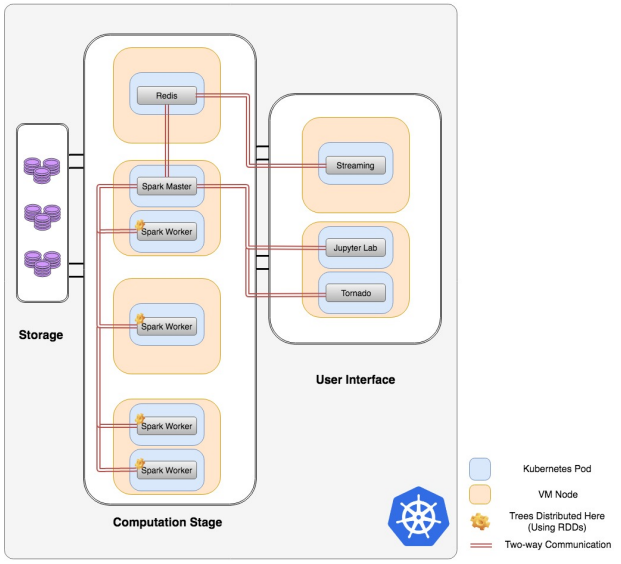


Figure 7: Simplified version of the current infrastructure used

Depending on the case, this configuration can be adapted to work with static data, where the training of the forest is done once and new data can be presented in the form of single point, bulk processing, or in the form of a stream. This can be accomplished with a queue manager and a broker to keep the cluster well balanced and within the load limits. For the case of streaming data only with a dynamic component, a forest can be trained initially and then it can evolve over time as new data is presented. A streaming version for the Isolation Forest algorithm is proposed in [13] and a revisited and

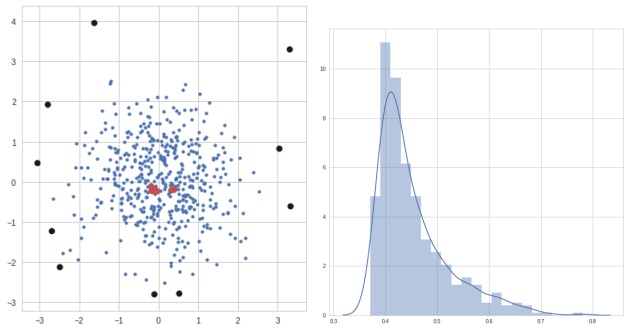
[5]https://www.docker.com

[6]https://redis.io

improved version is studied in [6] where the scale, shingle size, and pairwise distance are discussed. In our approach we can adapt our existing algorithm to work with different shingle sizes in order to improve the anomaly detection rate while reducing the false positives due to the naturally varying noise in the data. We can select a window in which the forest is recreated completely using newly added data or we can do it in stages where individual trees are updated and replaced in the forest incrementally. Our framework is flexible enough to consider all these, or even support multiple services at once.

4 RESULTS

We demonstrate a few example problems using Extended Isolation Forest and the technology infrastructure described above. We have already seen the score maps for the example of the blob shown in Figure 1. In Figure 8 we present the results of scoring each of the data points in this example using our infrastructure.



(a) Single 2D blob with anomalies (black) and nominal points (red).

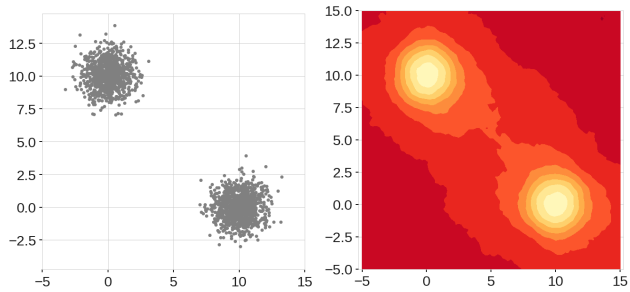
(b) Anomaly score Distribution

Figure 8: a) Shows the dataset used, some sample anomalous data points discovered using the algorithm are highlighted in black. We also highlight some nominal points in red. In b), we have the distribution of anomaly scores obtained by the algorithm.

We can clearly see the distribution of scores indicates anomalies are far less likely to occur. To look at a couple more examples, consider the datasets in Figures 9 and Figure 10. For a given data point, the higher score means it is more likely an anomaly. As it can be seen, the algorithm is very good at detecting anomalies even for hard cases such as the sinusoid where there is a repeating pattern in one of the features, figure 10. It is noteworthy that Extended Isolation Forest is a model-free algorithm. That is, it does not rely on having prior knowledge of the data at hand. Computation times and memory usage are low and scale linearly [9] and horizontally which works very well in our cloud infrastructure deployed using Kubernetes. The forest can easily be re-trained in the presence of new data, so even if we are trying to find anomaly in datasets that are changing over time, it is possible to adapt to this change in the data. Furthermore, the algorithm readily works with high dimensional datasets. In these particular examples we only show two dimensional data for visualization purposes.

4.1 Spark Configurations

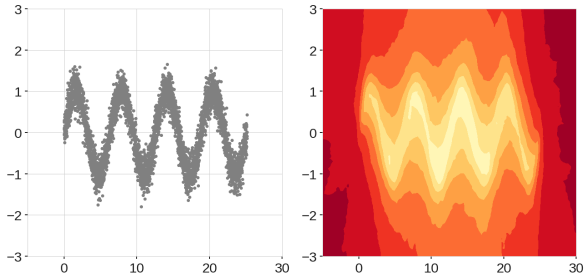
There are multiple variations of how one can parallelize the problem. We perform an experiment on a training dataset consisting of 4000



(a) Data points

(b) Anomaly score Map

Figure 9: Multiple blobs scored using Extended Isolation Forest. The algorithm captures the features of the dataset quite well. It even appropriately identifies lower anomaly scores directly in between the two blobs.



(a) Data points

(b) Anomaly score Map

Figure 10: Sinusoidal Data. The algorithm can capture the correct data structure even in the presence of complexities such as repeating values in one feature of the dataset.

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.

For comparison purposes, we have explored these variations against running the algorithm in serial in order to gain insight about improvements on computation time and the relative error we might incur.

We start an eight-node Spark cluster built on Kubernetes as described above. The parameter choices, such as number of trees were made with the consideration of the Spark cluster size in mind.

Case 1: Create 800 trees in a single core and run everything serially (training and scoring).

Case 2: Create 100 trees in each of the eight cores. Each core has access to the full dataset 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 forests are aggregated.

Case 3: Split the data for training among eight cores. In each core create 100 trees. Scoring is done similarly to Case 2.

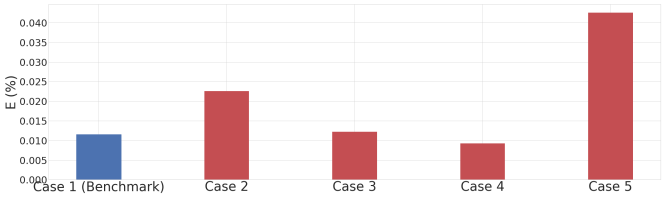
Case 4: Split the data for training among eight cores. Each core has 800 trees. Scoring is done similarly to the previous two cases. This naturally is expected to take the longest.

Case 5: Split the data for training among 8 cores. Score each partition (using only 100 trees) in one of the cores and simply collect data.

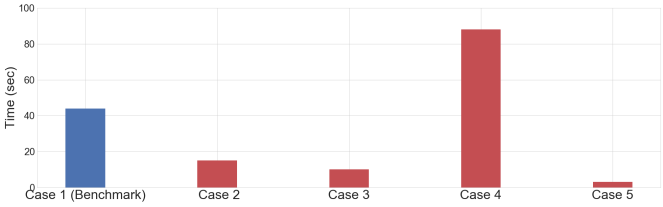
As mentioned before, the benchmark case is defined to be the case of serial computation. We compare the anomaly score for each data point, with S^{par} representing the score for parallel runs and S^{ben} representing the score for the benchmark case, in the metric measurement that follows:

$$E = \frac{100}{N}\left(\sum_{i=1}^{N}\left(\frac{S_i^{\text{par}} - S_i^{\text{ben}}}{S_i^{\text{ben}}}\right)^2\right)^{\frac{1}{2}} \quad (1)$$

Here N is the number of data points. Figure 11a shows the value of E for each case. As can be seen from this figure, the errors associated with each case is quite minimal (<0.05%). Note that the error associated with the benchmark case is not zero. This is because of the random nature of the Isolation Forest. There will always be a positive error comparing scores obtained from different runs of the same algorithm on the same data with identical parameters. Considering this fact, we can see that the parallel cases, specifically cases 3 and 4 are as accurate as the benchmark case. These values were obtained by averaging results from five different runs. The variance in the results was too insignificant in most cases to show on the figure. Figure 11b shows the comparison of computation times. We can see that significant time gain is possible. We note that reducing the number of trees affects accuracy of the results as expected, since the score does not converge fast enough with fewer trees. We notice that Case 3 is naturally the best option, since both the training and input data is divided, and the accuracy remains unaffected while the time is decreased significantly. We tried different sizes of the data and number of cores with similar results. Using Kubernetes and Spark we can horizontally scale our framework with ease to use Case 3 to handle any amount of training and input data.



(a) Average difference computed as shown in equation 1



(b) Total average time taken to run each case.

Figure 11: Average difference in time and accuracy for different ways of parallelizing the problem using MapReduce in Spark

In our particular example, and based on the error comparison and computation time plots, we concluded we could potentially choose case 3. In any specific application however, the same kind of benchmark analysis can be performed in order to choose the optimal strategy for parallelizing.

5 CONCLUSION AND FUTURE WORK

We aim to build a standalone batch and streaming based anomaly detection service that provides the user with a robust, scalable, and reliable service. This service at its core uses a model free and computationally inexpensive algorithm called Extended Isolation Forest. In our approach, we present an extension to the Isolation Forest which improves the performance of the algorithm, as well as a detailed technology stack from deployment to service delivery using containerized Spark cluster, orchestrated by Kubernetes accomplishing the following:

- An open source anomaly detection software package useful for a broad range of scientific cases.
- A ready-to-deploy infrastructure that is self contained, easily scalable, streamlined and robust for online and batch anomaly detection, with clear instructions, documentation and configuration files.
- Production services for scientific projects to analyze large datasets to detect interesting and exotic anomaly objects.

We presented some examples of anomaly detection using our Extended Isolation Forest. We also presented some benchmark evaluation of a test cloud infrastructure we are working with as well as the workflow and design of our anomaly detection framework. The algorithm fits the MapReduce paradigm quite well, and we plan on exploiting that fact in order to take advantage of full potential of Spark, run on a cloud platform with multiple scalable infrastructure and other forms of parallelism. At the end, we will provide a scalable platform that can be deployed anywhere with an operating Kubernetes cluster, making its access much easier and faster than ever possible.

The next steps will be to extend the current implementation to add other ancillary processing to deal with more complex and large multidimensional data as in the case for astronomy catalogs and images. For this we plan to use clustering algorithm to tackle the complex manifolds found in the data as well as analyze multiple datasets at a time using the same cloud technology. This will allow the analysis of the same data points under different criteria and training sets with less effort than needed by techniques used today. We will also add API access points to the framework so it can communicate with the user interfaces. With this API, web applications or Jupyter Lab can be used to search for anomalies, submit jobs, perform a parameter exploration, retrain the forest and more, these actions can also be wrapped and presented in a user and web friendly graphical interface. Lastly, we will add a broker interface such as Redis for alert notifications and message communication for applications with streaming data allowing for users to subscribe to upload, train, and score their own data.

The easy deployment, scalability, architecture, and robustness of our anomaly detection infrastructure deployed using containers and Kubernetes can prove useful in many scientific areas where an accurate, fast, and resilient anomaly detection is needed, without the need for deep technical knowledge.

REFERENCES

[1] 2018. Jupyter Lab. https://doi.org/doi:10.4231/D3F766898
[2] David Bernstein. 2014. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing* 1, 3 (2014), 81–84.

[3] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.

[5] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[6] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. 2016. Robust Random Cut Forest Based Anomaly Detection on Streams. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 2712–2721. http://dl.acm.org/citation.cfm?id=3045390.3045676

[7] Sahand Hariri and Matias Carrasco Kind. 2018. Extended Isolation Forest. *In preparation* (2018).

[8] Marc Henrion, Daniel J. Mortlock, David J. Hand, and Axel Gandy. 2013. *Classification and Anomaly Detection for Astronomical Survey Data*. Springer New York, New York, NY, 149–184. https://doi.org/10.1007/978-1-4614-3508-2_8

[9] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 413–422.

[10] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data* 6, 1, Article 3 (March 2012), 39 pages. https://doi.org/10.1145/2133360.2133363

[11] I. Nun, K. Pichara, P. Protopapas, and D.-W. Kim. 2014. Supervised Detection of Anomalous Light Curves in Massive Astronomical Catalogs. *The Astrophysical Journal* 793, Article 23 (Sept. 2014), 23 pages. https://doi.org/10.1088/0004-637X/793/1/23 arXiv:cs.CE/1404.4888

[12] Tiago Rosado and Jorge Bernardino. 2014. An Overview of Openstack Architecture. In *Proceedings of the 18th International Database Engineering & Applications Symposium (IDEAS '14)*. ACM, New York, NY, USA, 366–367. https://doi.org/10.1145/2628194.2628195

[13] Swee Chuan Tan, Kai Ming Ting, and Fei Tony Liu. 2011. Fast Anomaly Detection for Streaming Data. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. 1511–1516. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-254

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[15] Weijia Zhang and Xiaofeng He. 2017. An Anomaly Detection Method for Medicare Fraud Detection. (2017), 309–314.