FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Anomaly detection using Extended Isolation Forest
Student:	Bc. Adam Valenta
Supervisor:	Ing. Veronika Maurerová
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

Survey anomaly detection algorithms used to detect unusual patterns in data. Focus on the Extended Isolation Forest algorithm, where several decision trees are build to isolate anomalous data points. Implement the algorithm to the H2O-3 Open Source Machine Learning platform in the distributed Map/Reduce framework and utilize the Java Fork/Join framework for multi-threading. Test the functionality and scalability on toy problems, evaluate the performance on at least two anomaly detection datasets. Compare the performance with other open-source algorithms and discuss the results.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 21, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Anomaly detection using Extended Isolation Forest

Bc. Adam Valenta

Department of Applied Mathematics
Supervisor: Ing. Veronika Maurerová

May 28, 2020

Acknowledgements

I would like to thank Ing. Veronika Maurerová for supervision, leadership, smooth communication, and useful advices during the thesis. Many thanks also go to the H2O.AI company for giving me the opportunity to become a contributor to their product, as well as to the SSP portal for giving me the contact with H2O.AI. Furthermore, I would like to thank my partner MUDr. Julie Koskubová for providing me with an asylum and a quality working environment during COVID as well as for her patience. Last but not least, I would like to thank my family for their endless support during my studies and my cousin Vítězslav Grepl for proofreading this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 28, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Valenta, Adam. *Anomaly detection using Extended Isolation Forest.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Práce se zabývá různými typy algoritmů pro detekci anomálií, podrobně pak algoritmem Extended Isolation Forest. Extended Isolation Forest rozšiřuje svého předchůdce Isolation Forest. Původní Isolation Forest přináší zcela nový přístup k detekci, ale trpí zaujetím (bias) plynoucím ze způsobu, jakým vytváří stromy. Rozšířená verze algoritmu se tohoto zaujetí zbavuje úpravou větvení a původní algoritmus je jeho speciálním případem. Extended Isolation Forest je v rámci práce implementován do H2O-3 Machine Learning open-source platformy pro strojové učení. Základním požadavkem implementace je schopnost jejího spuštění na systému s distribuovaným výpočtem pomocí Map/Reduce knihovny.

Klíčová slova Detekce anomálií, Detekce outlierů, Novelty detection, Distribuovaný výpočet, Map/Reduce, Extended Isolation Forest, Open-source, H2O.AI

Abstract

The thesis deals with anomaly detection algorithms with a focus on the Extended Isolation Forest algorithm. Extended Isolation Forest generalizes its predecessor algorithm, the Isolation Forest. The original Isolation Forest algorithm brings a brand new form of detection, although the algorithm suffers from bias coming from tree branching. Extension of the algorithm removes the bias by adjusting the branching, and the original algorithm becomes just a special case. Extended Isolation Forest is implemented into the H2O-3 Machine Learning open-source platform. Implementation is required to run on a distributed computing system with a Map/Reduce library.

Keywords Anomaly detection, Outlier detection, Novelty detection, Distributed computing, Map/Reduce, Extended Isolation Forest, Open-source, H2O.AI

Contents

List of Figures

Introduction

In the age of processing and storage of rapidly increasing high volume information, commonly known as big data, it is necessary to have not only the means, but also methods, to analyze and evaluate this kind of information. It is important in the business sector, where correctly analyzing information could mean gaining a crucial lead in a highly competitive environment, but also in fields such as academia, healthcare, computer science and even space research. All these fields have an urgent need to analyze vast amounts of data to further their progress.

There are many ways to perform data analysis. One of them is the so-called anomaly detection or outlier detection. These two terms are equivalent, but only anomaly detection will be used in the thesis. Anomaly detection divides data into two types: normal (nominal) observations (samples, instances, distribution) and anomalies (outliers, novelties). Normal observation refers to data without any unusual characteristics; on the other hand, anomalies are data points that somehow differ from the normal observation – both in positive as well as negative sense. Negative anomalies in health care could signal a disease, in card transactions they could signal fraud. On the other hand in space data these could be new unknown objects and in the business sphere an important (new) customer.

The beginnings of anomaly detection date back to 19^{th} century statistics. Anomaly detection has been needed for a variety of domains and has developed separately in each domain. Hence many techniques use the specific nature of data in a particular field or industry. Before the era of big data there was no intention for a general technique of anomaly detection. With the evolution of data analysis, researchers began to study the specific techniques, defined the nature of the data, and tried to develop a general anomaly detection algorithm that would be scalable through multiple domains.

Goal and Motivation

Most of the existing anomaly detection algorithms are based on an assumption that precise knowledge of the normal observation behavior leads to a good understanding of anomalies. In other words, the algorithms are optimized to find a normal observation, not the anomalies themselves.

In 2008 several researchers recognized these approaches and developed the Isolation Forest (IF) algorithm, one of few algorithms that is optimized primarily for detection of anomalies instead of detection of normal observations. IF is based on an assumption that anomalous points are few and different, hence it is easy to differentiate and isolate them from normal points. IF builds hundreds of decision binary trees and studies the mean length path of the point through all the built trees. The assumption that anomalies are closer to the root implies a shorter path through the trees. On the contrary, normal points are deeper in the trees, and hence the path is significantly longer for them.

The thesis deals with the Extended Isolation Forest (EIF) algorithm that was presented in 2018 as an improvement of the Isolation Forest algorithm. IF's performance is promising, though it does suffer from bias due to the method it branches binary decision trees. EIF implements a new technique for branching, which leads to the improvement of the anomaly score distribution and mitigates this bias. The EIF uses a new hyperparameter "extension_level" in order to allow for ordinary IF to run inside the EIF algorithm. The EIF algorithm was developed in Python and currently there is no other known implementation of the algorithm that would be suitable for a production environment or for processing of big data.

The goal of this thesis is to implement the Extended Isolation Forest algorithm in Java. The algorithm will be implemented into the H2O-3 Machine Learning open-source platform developed by the H2O.AI company. H2O-3 is one of the company's products for distributed Machine Learning, written in Java, using the Map/Reduce framework for cloud computing. Therefore the implementation of the EIF in Java will be tested, and its results compared to the only known implementation in Python. Furthermore, the algorithm's Java implementation will be compared to the existing open-source implementations of Isolation Forest in various libraries, including H2O.AI's.

The motivation is to create the first production-ready, well tested, and fully supported implementation of Extended Isolation Forest with the possibility to scale the algorithm for big data. This will allow for a significantly wider usage of the EIF, as the model will be implemented into a widely used platform, thereby democratizing the EIF for the masses who will be able to use implementation for their experiments, ranging from small with MBs of data to large ones with TBs of big data.

Chapter 1

Anomaly detection

In the beginning, it is necessary to define why anomalies should be detected and what to do with them. The motivation for researchers to find anomalies is different across various domains; however, the common part is *"interestingness"* of anomaly points and their real-life relevance [1]. Anomalies should be identified for further study as this knowledge could be used to create a more robust model that can take into account the possibility of anomalies' presence even without being interested in the specific anomalies [2].

> *"The identification can lead to (i) rejection (removal) of spurious data; (ii) recognition of important new information or even (iii) revision of the model describing the data by incorporating allegedly outlying elements; (iv) refinement of the experimental setup."* [2]

As was already written, the term anomaly is equal to *outlier*, and the thesis uses anomaly, but there are also two related terms. First is *"noise"*.

> *"Noise can be defined as a phenomenon in data which is not of interest to the analyst, but acts as a hindrance to data analysis. Noise removal is driven by the need to remove the unwanted objects before any data analysis is performed on the data."* [1]

Thus, in that respect, noise cannot provide any value and thereby is excluded from anomaly detection.

Second is *"novelty"*. Novelty could be considered the same as an anomaly, the difference is, let's say, business value of detected point. Anomaly, in its nature, is an interesting point different from normal points with no chance to become a normal point. Novelty, on the other hand, is a point (in most cases several points), firstly detected as an anomaly, but after domain expert's

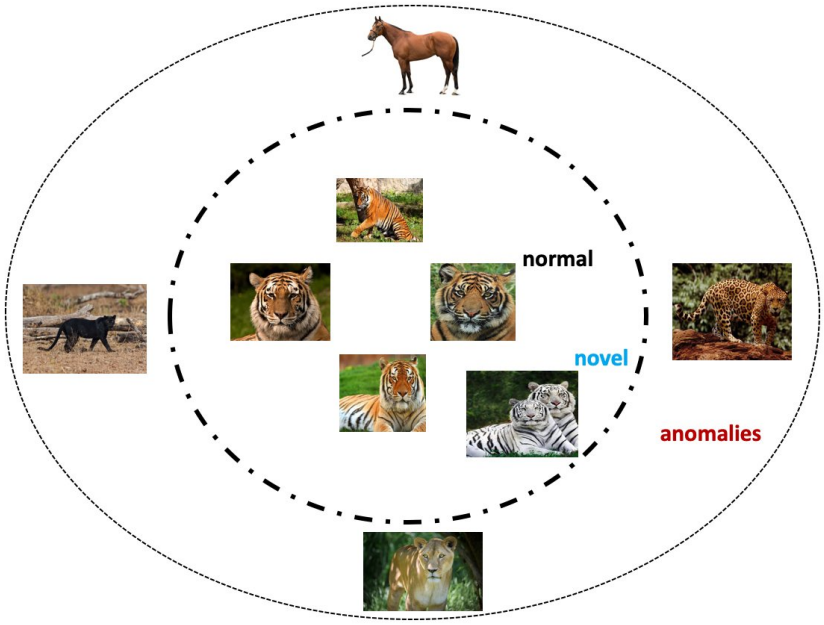


Figure 1.1: Novelty detection [3]

intervention, the point becomes a normal observation. From the anomaly detection algorithms point of view, novelty and anomaly have the same characteristic, but novelties are subsequently included in the normal distribution if the model's rebuild. The [3] says that in case of the implementation, novelty has a lower threshold than the threshold for the anomaly. [1][2]

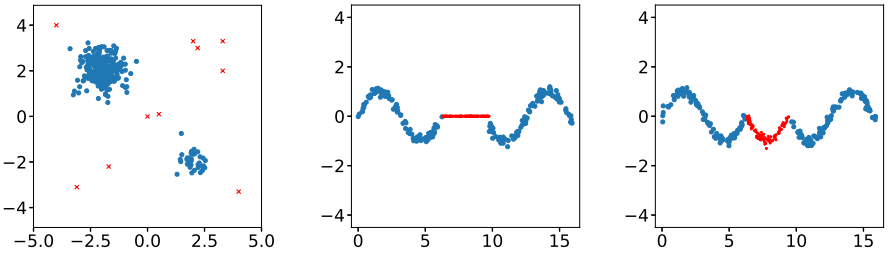
The reason why anomaly detection is needed is quite clear. However several different points of view exist for anomaly detection or the anomalies itself. It is a remnant of previous separated research in a single domain without the focus on generalization. The [1] studied numerous papers and provides a common point of view for anomaly points and anomaly detection techniques' typology. Paper [3] builds on the same definition. The thesis uses this definition - list of terms is in the section below.

1.1 Types of Anomalies

Point Anomalies. Instance significantly different from the rest of the data. A simple and intuitive definition of an anomaly.

Contextual Anomalies. It is a Point Anomaly only when context of the data is considered. For example, mean temperature of 5°C is anomalous in July but not in January because the context of a month is taken into account.

Figure 1.2: Anomalies



(a) Point Anomalies (b) Collective Anomalies (c) Contextual Anomalies

Another example of this can be usage of electricity - in December it is high because of the Christmas lights and this could be considered an anomaly with respect to the whole scope of data. Nevertheless, it is a normal observation if the context of Christmas is taken into account.

Collective Anomalies. Collection of related instances is anomalous only when observed as a collection; the individual point in the collection may not be an anomaly. For example, suppose data with the records of a patient's breath during a night's sleep with 1s samples. Point "not breathing" is a normal instance if it appears as a single observation. Collection of twenty "not breathing" points is an anomaly.

In the rest of the thesis, the term anomaly always refers only to point anomalies. The other types were explained for completeness and are not covered by the thesis.

1.2 Data for Anomaly Detection

Anomaly detection deals with all sorts of data. Images, time series or tabular data are very common. For the purposes of this thesis only data in a tabular structure, meaning rows and columns, will be considered. Data preprocessing is up to the user. Algorithms expect that data type is a real number. A row is also referenced as variable x or point, observation, instance, and columns could be addressed as features. It is possible to mark rows as either normal or as an anomaly. These marks are called labels. Variable *"N"* stands for number of rows and variable *"P"* is number of columns (Features).

To sum up,

- $DATA \in IR^{N,P}$,
- a row of $DATA$ is $x \in IR^{P}$,
- N is number of rows, P is number of features.

The data is used to create an anomaly detection model. Model creation is also called building, training, or learning. The created model is then used to determine whether the instance is an anomaly or not. This process can be referenced as an evaluation, testing, or prediction.

1.3 Output of the Anomaly Detection

The difference among algorithms used for anomaly detection, besides used techniques, is the output. There are techniques that give a two value classification, normal point, and anomaly; typically these are algorithms that come from the clustering family (2). The second possible output is an anomaly score and user has to define a threshold for an anomaly. The score is most probably a continuous variable between 0 and 1 or between -1 and 0 (depending on the implementation). The thesis always considers range $[0, 1]$ where anomaly score closer to zero means normal point and score closer to 1 stands for an anomaly. Note that among the multiple implementations of the same algorithm with anomaly score output, the definition of normal and anomalous point value may differ. Reading of documentation is essential in all cases. [1][4][5]

Indeed, the similarity to common data science typology of regression and classification is obvious. The difference here is that output should always be labeled as either normal or as an anomaly. In the first case, the label comes straight from the output. In the second case there is an opportunity to study the distribution of the anomaly score, with the user deciding what is and what is not an anomaly.

1.4 Task for Anomaly Detection

Research in [1] states that anomaly detection deals with:

- Definition of the normal region for every possible normal observation without a precise boundary between normal and anomaly observations.
- Anomalies could be a result of fraudulent action, and those actions are most likely adapted to behave very similarly to normal actions.
- Normal observations can evolve, and models become increasingly outdated as time passes.
- The nature of anomalies differs across domains, a general algorithm should take this into account.
- No availability of labeled data for model evaluation.

The literature also introduced two common problems for anomaly detection based on the nature of anomalies:

Masking. It happens when given data contains anomaly clusters. Anomaly is not detected because of the close presence of other anomalies. Hence the anomaly's presence is masked by the close points. [4][6][7]

Swamping. Refers to the situation when normal points are considered to be an anomaly merely because the normal point is too close to an anomalous point. [4][6][7]

1.5 Types of Anomaly Detection Techniques

Both studies [1] and [3] provide a taxonomy of anomaly detection techniques. First one explores traditional data mining techniques while also covering Deep Learning (Methods based on artificial Neural Networks (NN)), second study focuses only on Deep Learning (DL). Furthermore both studies sorted detection techniques from two points of view. Firstly the given data; and secondly the algorithm technique. Note that the sorting is not exclusive. The techniques can diverge between the types, but the goal is to define the assumptions of the algorithms.

1.5.1 By the input

The simple and most common typology of any data mining technique is by the input given to the algorithm. Anomaly detection is not an exception. The techniques can be divided into three classes for data mining and Deep Learning. The [3] study also provides two more classes based on the combination of DL and data mining. Suppose a data with only two labels, normal and anomaly:

Supervised. Both labels are provided, but the data is highly unbalanced. There are countless normal observations and few anomalies among them. This type of data is rare, and the technique is the same as building a predictive model with highly unbalanced data.

Semi-Supervised. Only one label is provided, most likely for the normal observation, but having a label only for anomalies is also possible in some cases. The techniques operating in this class most likely build a model for normal observation and use it to identify anomalies.

Unsupervised. No labels are provided. Most widespread type of anomaly detection. The essential assumption for anomaly detection is that normal observations are far more frequent than anomalies. A technique without this assumption is called binary clustering.

Consider the nature of anomaly detection. Only Semi-Supervised and Unsupervised types of algorithm is taken into account. Supervised techniques are excluded from the scope since it is data mining with highly unbalanced data. The (Extended) Isolation Forest belong to the Unsupervised section.

The two additional techniques proposed by the [3] are:

Hybrid deep anomaly detection. Deep Learning model is used as a feature extractor to learn robust features. The features are the input into any Machine Learning model. The algorithms based on hybrid detection are scalable to high dimensional data.

One-class Neural Networks. One-class Neural Networks combine the ability of deep networks to extract a progressively rich representation of data along with the one-class objective. The model simultaneously trains a deep neural network, and optimizes data-enclosing. Anomalies do not contain common factors and so data-enclosing fails on them. A disadvantage is the long time required to train the model.

1.5.2 By the nature of used algorithm

The typology here comes from [1]. Section 1a in the list below is extended with the possible Neural Networks architectures presented in the [3] survey. It has to be kept in mind that combining both studies is not easy nor straight forward. For every class, besides 1a, it is possible to find a representative from the Neural Networks family as a standalone or combined with a traditional Machine Learning algorithm. The purpose of the list is to only get an idea what kind of algorithms can be used. It also serves as a starting point to get more information in [1] and [3]. Two algorithms from point 2, one from SVM and one from Deep Learning will be presented in the following section.

The (Extended) Isolation Forest algorithm brings a new anomaly detection technique, and that is the reason why it cannot be assigned to any of the classes. IF and EIF build an ensemble of trees similar to the Random Forest algorithm, thereby new technique *"Ensamble Based"* would be a good proposal. The Scikit-learn also includes Isolation Forest implementation into the ensemble package [5].

1. Classification Based
    a) Neural Networks Based
        i. Deep Neural Networks
        ii. Spatio Temporal Networks (STN)
        iii. Sum-Product Networks
        iv. Word2vec Models
        v. Generative Models
        vi. Convolutional Neural Networks
        vii. Sequence Models
        viii. Autoencoders
    b) Bayesian Networks Based
    c) Support Vector Machines Based
    d) Rule Based
2. Clustering Based
3. Nearest Neighbor Based
    a) Using Distance to k^{th} Nearest Neighbor
    b) Using Relative Density
4. Statistical
    a) Parametric Techniques
        i. Gaussian Model Based
        ii. Regression Model Based
        iii. Mixture of Parametric Distributions Based
    b) Non-parametric Techniques
        i. Histogram Based
        ii. Kernel Function Based
5. Information Theoretic
6. Spectral

1.6 Anomaly Detection Algorithms

In this section, several anomaly detection algorithms are described. K-means was chosen for it is simple and easy to understand. DBSCAN further develops the clustering by considering anomalies. SVM brings a probability estimation into the anomaly detection task. Last but not least, the Autoencoders are presented as a representative of NN based techniques. Note that this section is only a brief introduction to these algorithms. The details for each are in the mentioned sources.

1.6.1 K-means

Following algorithm from a cluster-based family is an adjustment of K-means. It goes without saying that K-means is a widely known clustering algorithm. The algorithm creates "K" similar clusters of data points. K-means uses the critical assumption that normal instances belong to "K" several groups and points that fall outside of these groups could be marked as potential anomalies. In K-means implementations, points called centroids are computed to define the core of the "K" cluster, and the points closer to one of the centroids are marked as the class of the particular point. The notorious algorithm can be easily adjusted for anomaly detection by adding a threshold for the distance between the centroid of normal observations and the analyzed point, as can be seen in Figure 1.3. There are different methods for the classification of anomalies, but all of them use a sort of the mentioned distance to the centroid. The output of this algorithm is the classification of points as either normal or as anomalies. [8][9][10]

The significant advantage of this approach is that it is simple and the anomaly detection is easy to understand. Nonetheless this algorithm has disadvantages, which should be critically considered prior to deciding to use the algorithm in a production deployment. Firstly, choice of the "K" hyperparameter [10]. There are techniques to choose K, but it is difficult to set a number of clusters. This is especially true with high dimensional data, where visualization possibilities are limited. Second problem of the K-means is that it is only suitable when clusters can be expected to have relatively regular shapes [11]. Thirdly, the anomalies themselves affect the centroid's computation. This should not be a significant problem since anomalies are rare, but it is a disadvantage compared to the other algorithms [10][11]. Fourthly, since there is no lower limit for the number of points in a cluster, it is possible to create a cluster of perfectly defined anomalies with an unknown "K" [11].

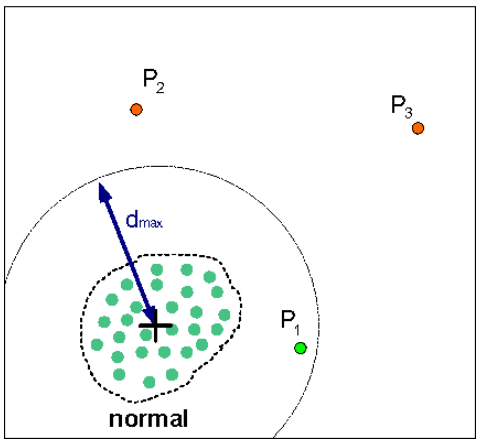


Figure 1.3: K-means - anomaly detection [8]

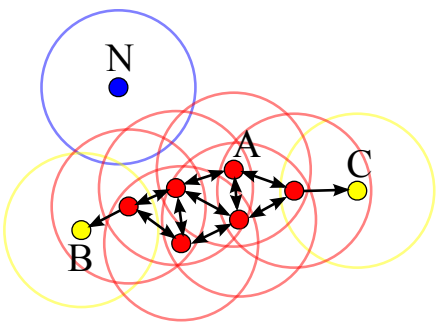
1.6.2 DBSCAN

(DBSCAN) is another example of a cluster based algorithm that considers the possibility of anomalies in its application. It requires two parameters:

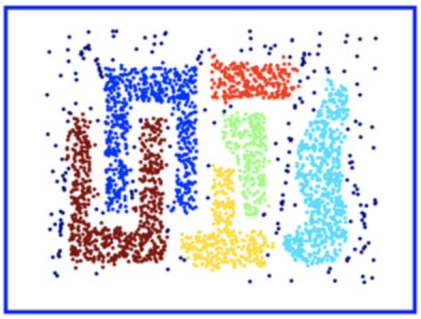
- neighbor radius value ε (eps) with arbitrary measure of distance,
- minimum number of points required to form a dense region (minPts).

The algorithm presented in [12] paper defines a terminology for the points in the data. This thesis abstracts the terminology and gives a high-level interpretation of the algorithm. See picture 1.4a. In essence the algorithm starts with a random point (A), point is considered to be in the same cluster when it has a value higher than "minPts" neighbors within the radius ε (including the query point). The anomalous point (N) is the point with a value less than "minPts" neighbors within ε radius. This process continues for each point until no undecided points exist. It should be added that the DBSCAN algorithm was originally presented in 1996 by [13] and became a core idea for future variations such as HDBSCAN*, OPTICS or LSDBC. The same happened with the Isolation Forest algorithm - specifically, DeepIF [14], iForestASD [15], Extended Isolation Forest [16] and several more.

Figure 1.4: DBSCAN visualization



(a) DBSCAN [12]



(b) DBSCAN - clustering [17]

DBSCAN is suited to separate high-density clusters from low-density clusters (1.4b). On the other hand, the algorithm struggles with clusters of varying densities. Although DBSCAN separates data with contorting clusters, it suffers when data has too many dimensions. Also, algorithm's output can vary depending on if it starts with a random choice of the first point like K-means, but the number of anomalies has no effect on computing the cluster point. Last but not least, the purpose of DBSCAN is to precisely estimate the cluster of normal points while considering possible occurrence of anomaly points. However, according to [1] DBSCAN is not optimized to find an anomaly. [17]

1.6.3 Support Vector Machine Extension For Anomaly Detection

SVM is a kernel technique typically associated with supervised learning; however, there are adjustments of the original algorithm for unsupervised learning suited for anomaly detection [9]. Connection of kernel-based techniques and density-based approaches is well summarize in the following citation:

> *„Clustering algorithms are further examples of unsupervised learning techniques which can be kernelized. An extreme point of view is that unsupervised learning is about estimating densities. Clearly, knowledge of the density of P would then allow us to solve whatever problem can be solved on the basis of the data.“* [18]

The algorithm presented in [18] computes binary functions, which is supposed to capture regions in an input space where is the probability density of normal points. In particular, a function such that most of the data will live in the region where the function is nonzero. The goal is to bind the probability that an anomaly point from the same distribution will lie outside of the estimated region by a certain margin.

1.6.4 Autoencoders

Autoencoders or also Replicator Neural Networks (RNN) represent data inside multiple hidden layers by iterative reconstruction of the input data. Well trained autoencoders can precisely reconstruct any given normal instance, but for anomalies they produce significant errors during reconstruction. The points that generate high amount of errors are considered anomalies. [3]

> *„The choice of autoencoder architecture depends on the nature of data, convolution networks are preferred for images while Long short-term memory (LSTM) based models tend to produce good results for sequential data.“* [3]

Autoencoders are simple and effective NN for anomaly detection, but the performance is degraded if autoencoders are dealing with training data containing a high amount of noise [3]. The paper [19] provides a way to create Robust Deep Autoencoder (RDA)s inspired by Robust PCA. The algorithm presumes that real word data is noisy and deals with this via a method based on splitting the input data into two classes. One class contains data that can be easily reproduced, and the second contains anomalies and noise. After splitting, the autoencoder is trained on clean, noise-free data and thus provides a robust estimation of the normal observation. The details of the method itself are in [19] paper.

The authors compare Robust Deep Autoencoder to the Scikit-learn implementation of Isolation Forest [5] on a noisy version of the MNIST data and conclude that RDA improves the F1-score by approximately 73%. The reason why the IF performs worse might be due to the 28x28 pixel image, which leads to 784 features. The Isolation Forest could perform better when combined with a feature extraction method.

Chapter 2

Isolation Forest

Algorithm Isolation Forest (IF) proposes a new technique of anomaly detection different from all previous ones. The algorithm builds on the assumption that anomalies are "few and different". Instead of profiling a normal distribution and measuring the distance of each point to the normal points, the algorithm instead isolates anomalies from the rest of the points and measures their difference using the Isolation Tree (iTree). This approach, together with an ensemble of Isolation Trees and the sub-sampling method, leads to a model with low memory requirements as well as low logarithmic complexity of the training and evaluation stages. [4]

2.1 Concept of Isolation

In this context, the term isolation means "separating an instance from the rest of the instances". Suppose a data-induced Binary Decision Tree with an anomaly present. The assumption "few and different" implies anomalies are decided closer to the root, and normal points are deeper in the tree. The binary tree is built to isolate all the points and measure their individual Path Lengths from the root. The following definitions comes from the paper [4].

Definition 1: Isolation Tree (iTree). Let T be a node of an Isolation Tree. T is either an external-node with no child, or an internal-node with one test and exactly two daughter nodes (Tl, Tr). A test consists of an attribute q and a split value p such that the test $q < p$ divides data points into Tl and Tr.

Definition 2: Path Length $h(x)$ of a point x is measured by the number of edges x traverses an $iTree$ from the root node until the traversal is terminated at an external node.

Definition 3: iForest of size t is an ensemble of t $iTrees$.

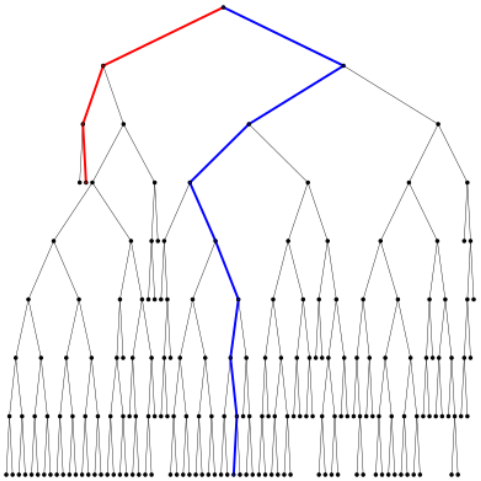


Figure 2.1: Isolation Tree [16]

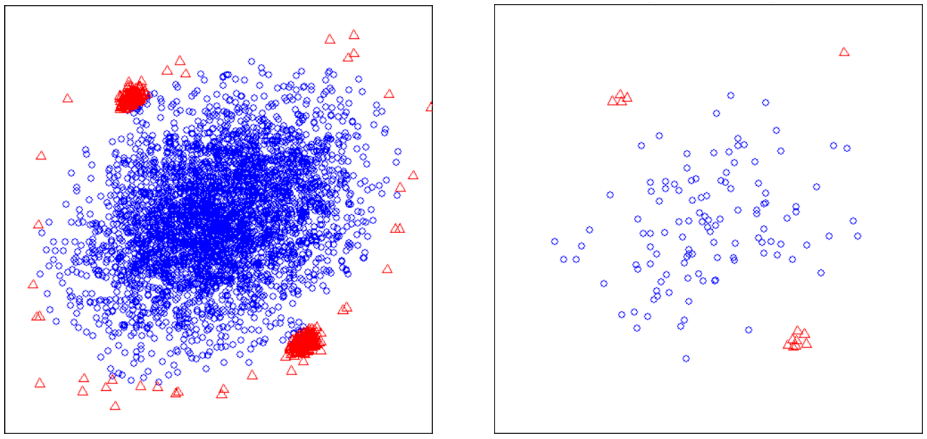
Suppose data is in the format from section 1.2. In short, *DATA* are matrices of real numbers of a dimension NxP, where N is number of rows and P is number of features. In training stage, the Isolation Forest algorithm builds an ensemble of $iTree$ over *DATA*. In the evaluation stage, for any value x the mean $h(x)$ in ensemble of $iTree$ is computed. In the following sections the average $h(x)$ is used to calculate the anomaly score.

2.2 Concept of sub-sampling

Problem of *masking* and *swamping* was introduced in section 1.4. The goal of IF is to provide a robust model that mitigates the effects of these two obstacles. *Masking* is introduced when anomaly cluster is large and dense, as this also increases the number of splits in the IF. *Swamping* also increases the number of splits required to separate anomalies, which leads to a high average $h(x)$. [4]

> *"Note that both swamping and masking are a result of too many data for the purpose of anomaly detection."* [4]

The nature of IF provides an opportunity to avoid *masking* and *swamping* by employing the sub-sampling method. Sub-sampling refers to a random selection of rows of data without replacement. The benefit for IF's anomaly detection is data size control, which helps the algorithm to isolate the points better, and it also improves the diversification of each $iTree$, since the ensemble is created. The basic assumption is that anomalies are "few and different". The probability of even selecting a dense cluster of anomalies is lower than



(a) Original Sample (4096 instances) [4] (b) Sub-sample (128 instances) [4]

selecting a cluster of truly normal instances. Finally, the sub-sample of *DATA* is created before each *iTree* in the *iForest* is built. [4]

The impact of the sub-sampling size could also be studied. In the case of IF, there is an upper bound for the sub-sampling size to get the same detection performance without increasing memory requirements. The size is empirically determined to the 256 rows. [4]

2.3 Anomaly score computing

The output of the IF algorithm is an anomaly score. Considering the user-defined threshold the IF algorithm labels individual points either as normal or as an anomaly. In short, the anomaly score is average $h(x)$ in $iForest$ normalized by the average path of unsuccessful searches in a Binary Search Tree (BST). In the following part, the individual components of the anomaly score formula are presented.

Average $h(x)$ of the unsuccessful search in BST for the data set of size i is:

$$c(i) = \begin{cases} 2H(i-1) - \frac{2(i-1)}{i} & \text{for } i > 2 \\ 1 & \text{for } i = 2 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where:

$H(.)$ = harmonic number estimated as $ln(.) + 0.5772156649$ (Euler's constant [20]),

As is written in the section Isolation Forest Algorithm 2.4.1, the height of $iTree$ is limited so as to manage memory requirements. Formula (2.1) is used to estimate the tree height in cases, where $iTree$ is not able to isolate the point. This is done especially for dense clusters of normal points.

The anomaly score formula is:

$$s(x, N) = 2^{\frac{-E(h(x))}{c(N)}} \tag{2.2}$$

where:

x = any row in the *DATA*
N = number of rows in the *DATA*
$E(h(x))$ = mean $h(x)$ in ensemble

The anomaly score is interpreted as follows:

- if instances return s very close to 1, then they are definitely anomalies,
- if instances have s much smaller than 0.5, then they can be quite safely regarded as normal instances,
- if all the instances return s around 0.5, then the entire sample does not have any distinct anomalies.

2.4 Isolation Forest Algorithm

The hyperparameters of the IF model are:

t = number of trees,
ψ = sub-sampling size.

The algorithm is split into two stages. First is the Training stage where the Forest is created. The second stage is the Evaluation stage, which puts a given point into each tree and provides an average $h(x)$ of the point. Complexity of the IF algorithm is the same for both stages:

$$\mathcal{O}(t\psi log\psi) \tag{2.3}$$

2.4.1 Training stage

The training stage performs sub-sampling and builds an ensemble of $iTree$s. Each $iTree$'s height is limited by its $ceiling(\log_2 \psi)$, which is approximately the average height of BST for size of given data [4]. Each $iTree$ focuses on points that have a shorter path than average. The algorithm for training is separated into two functions. Recursion is used in Algorithm 2 for building the $iTree$. The output of training stage is an $iForest$ prepared for scoring of each given point.

Algorithm 1 $iForest(X, t, \psi)$

Require: X - input data, t - number of trees, ψ - sub-sampling size
Ensure: a set of t $iTrees$

Initialize $Forests$
set height limit $l = ceiling(\log_2 \psi)$
for $i = 1$ to t **do**
    $X' \leftarrow sample(X, \psi)$
    $Forest \leftarrow Forest \cup iTree(X', 0, l)$
end for
return $Forest$

Algorithm 2 $iTree(X, e, l)$

Require: X - input data, e - current tree height, l - height limit
Ensure: an iTree

if $e \geq l$ or $|X| \leq 1$ **then**
    return $exNode\{Size \leftarrow |X|\}$
else
    let Q be a list of attributes in X
    randomly select an attribute $q \in Q$
    randomly select a split point p from max and min $values$ of attribute q in X
    $X_l \leftarrow filter(X, q \leq p)$
    $X_l \leftarrow filter(X, q > p)$
    return inNode$\{Left \leftarrow iTree(X_l, e + 1, l)$,
        $Right \leftarrow iTree(X_r, e + 1, l)$,
        $SplitAtt \leftarrow q$,
        $SplitValue \leftarrow p\}$
end if

2.4.2 Evaluation stage

The output algorithm of the evaluation stage is $h(x)$ of the given point. Average $h(x)$ in $iForest$ is computed and handed over to the anomaly score formula (2.2). In Algorithm 3 the formula (2.1) is used to estimate $h(x)$ for the cases, where IF is not able to isolate the points.

Algorithm 3 $PathLength(\vec{x}, T, e)$

Require: $\vec{x}$ - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
Ensure: path length of $\vec{x}$
1: **if** T is an external node **then**
2: **return** $e + c(T.size)$ {$c(.)$ is defined in Equation (2.1)}
3: **end if**
4: $a \leftarrow T.splitAtt$
5: **if** $x_a \leq T.splitValue$ **then**
6: **return** $PathLength(\vec{x}, T.left, e + 1)$
7: **else** $x_a > T.splitValue$
8: **return** $PathLength(\vec{x}, T.rigth, e + 1)$
9: **end if**

2.5 Solution to high dimensions

Isolation Forest algorithm suffers from the dimensionality curse. The paper [4] designs a solution based on combining the Isolation Forest algorithm with a feature selection method. Sub-set of features based on Kurtosis statistical test is selected from a sub-sample of data before each $iTree$ is built. The experiment in [4] shows the time advantage of IF due to low processing requirements on high dimensional data.

2.6 Advantages and Disadvantages of the Isolation Forest Algorithm

Isolation Forest algorithm uses a different perspective to detect anomalies. Rather than waste resources on precise defining the normal distribution, it instead focuses on isolation of anomalies. The defined $iTree$ structure is similar to BST, thanks to which the memory and computation needs are low. The empirical evaluation in [4] shows that Isolation Forest performs significantly better than a near-linear time complexity distance-based method. However, solutions based on Deep Learning have a better performance than IF [3]. Unlike the K-means adjustment that needs to have the "K" hyperparameter defined, there is no need for input of number of clusters in IF. This leads

to less maintenance of the production model. In case that the production data evolves, there is a possibility to set a threshold for novelties as [3] suggest. On the other hand, there is no possibility for continuous updating of the model - hence if data significantly evolves the model has to be re-built. Furthermore the IF algorithm lacks the *minPts* advantage of the DBSCAN algorithm, which allows create lower bound for a number of points required to form a dense region. Nevertheless IF is optimized to find an anomalies, not for defining the normal distribution.

The distribution of the anomaly score should also be mentioned. Anomaly score suffers from bias caused by the way the $iTree$ are split. The [16] proposes a simple but ingenious solution to get rid of this bias and builds a more robust model with the same performance. Hence it can be concluded that IF is a special case of the general isolation method.

Chapter 3

Extended Isolation Forest

Extended Isolation Forest (EIF) is an algorithm for unsupervised anomaly detection based on the Isolation Forest algorithm. The extension lies in the generalization of the Isolation Tree branching method. Original IF branching provides slicing only parallel to one of the axes. EIF's branching method allows for slicing of the data by using hyperplanes with random slopes.

Motivation for the Extended Isolation Forest algorithm was a study of anomaly score given by IF on toy 2-D data. As can be seen in Figure 3.1, since the data is generated from a 2-D Gaussian $\mathcal{N}((0,0),(1,1)$ the IF should estimate the lower anomaly score close to the point $(0,0)$ and an almost circular and symmetric pattern with increasing values of anomaly score as it radially moves outward. Another case is shown in Figure 3.2. Occurrence of ghost clusters is evident. This could lead to false positive anomaly detection since the threshold is used to find anomalies. [16]

If the splitting of each point is visualized (3.3) and connected with the score maps, it can then be inferred that it is due to the split method. The way BST branching creates lines parallel to one of the axes, which in turn causes the bias. [16]

3.1 Generalization of Isolation Forest

The visualization of the BST splitting leads to the idea that the branching should be more randomized and not merely focused on the lines parallel to the axes. Paper [16] proposes two methods for the generalization. First is suitable for solutions where it is not possible to change the IF algorithm. It proposes the rotation of the sub-sample before each $iTree$ is built, thereby "averaging out" the bias. Second method introduces an adjustment for IF algorithm which leads to a complete generalization of the IF method, which is called the Extended Isolation Forest.

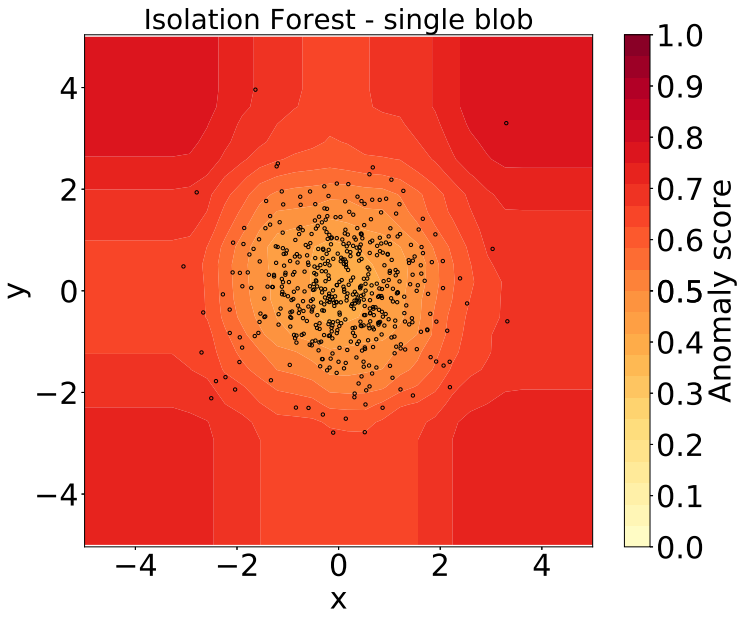


Figure 3.1: Isolation Forest - single blob

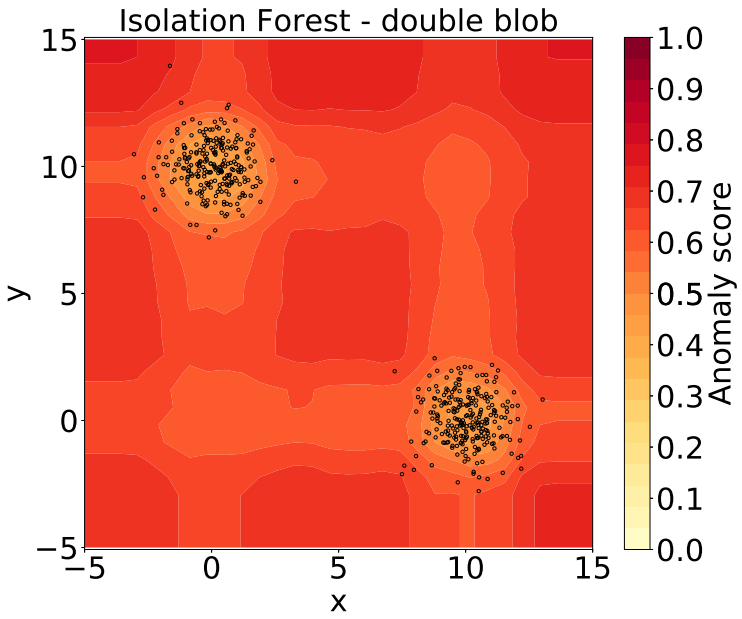


Figure 3.2: Isolation Forest - double blob

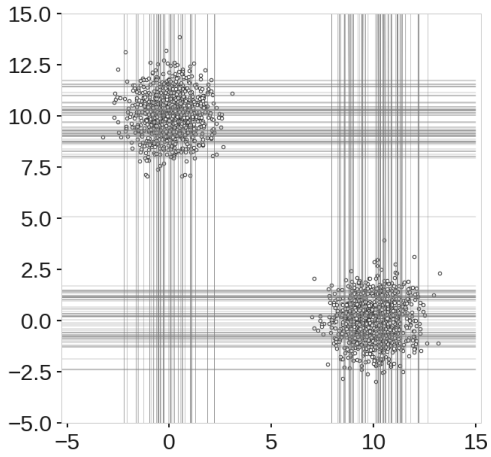


Figure 3.3: Splits [16]

3.1.1 Rotated Trees

Original IF creates a sub-sample of data before each $iTree$ is built. The bias can be averaged out if each $iTree$ is built on a rotated sub-sample. In such case the bias still exists, however it is different for each $iTree$ and the bias is averaged out. Despite this leading to improved results, the method is still insufficient for the following reasons:

- Each $iTree$ still suffers from the bias (though the bias is different for each tree).
- Extra information about the angle of the rotation for each tree needs to be saved.
- If the data lack symmetry, this method is not sufficient.
- It is not suitable for large and high dimensional data.

3.1.2 Branching adjustment

The cause of the bias is that branching is defined by the similarity to BST. At each branching point the feature and the value are chosen; this introduces the bias since the branching point is parallel to one of the axes. The general case needs to define a random slope for each branching point. Instead of selecting the feature and value, it selects a random slope $\vec{n}$ for the branching cut and a random intercept $\vec{p}$. The slope can be generated from $\mathcal{N}(0, 1)$ Gaussian distribution, and the intercept is generated from the uniform distribution with bounds coming from the sub-sample of data to be split. [16]

At this point, a new generalization hyperparameter, *extensionLevel*, is introduced. The function of *extensionLevel* is to force random items of $\vec{n}$ to

be zero. The *extensionLevel* hyperparameter value is between 0 and $(P-1)$. Value of 0 means that all slopes will be parallel with all of the axes, which corresponds to Isolation Forest's behavior. The higher number of *extensionLevel* indicates that the split will be parallel with *extensionLevel*-number of axes. The full-extension means *extensionLevel* is equal to $P-1$. This indicates that the slope of the branching point will always be randomized. The [16] recommends to use a completely extended EIF. A lower extension is suitable for a domain where the range of the minimum and maximum for each feature highly differs (for example, when one feature is measured in millimeters, and the second one in meters). [16]

3.2 Extended Isolation Forest Algorithm

This section presents the pseudo-code of Branching adjustment 3.1.2 from [16]. The first method of Rotated Trees 3.1.1 is not included in any pseudo-code nor in any further implementation. The algorithm is also separated into two stages. First for $iForest$ building and second for the $h(x)$ computation.

The hyperparameters of the EIF model are:

t	= number of trees,
ψ	= sub-sampling size,
$extensionLevel$	= value in range $[0, P-1]$; where P is the number of features.

3.2.1 Training stage

The first part of the algorithm is known from the original IF. There is no difference in the pseudo-code. The change lies in Algorithm 5, concretely in lines 4-8, where the branching is adjusted accordingly. Line 9 shows that the $iTree$ contains vectors and no single value. The size of the vector is the number of features. Thus the memory requirements are not significantly increased as the number of features remains significantly lower than the number of rows.

Algorithm 4 $iForest(X, t, \psi, extensionLevel)$

Require: X - input data, t - number of trees, ψ - sub-sampling size
Ensure: a set of t $iTrees$

1: **Initialize** $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: **for** $i = 1$ to t **do**
4: $\quad X' \leftarrow sample(X, \psi)$
5: $\quad Forest \leftarrow Forest \cup iTree(X', 0, l, extensionLevel)$
6: **end for**
7: **return** $Forest$

Algorithm 5 $iTree(X, e, l, extensionLevel)$

Require: X - input data, e - current tree height, l - height limit
Ensure: an iTree
1: **if** $e \geq l$ or $|X| \leq 1$ **then**
2: **return** $exNode\{Size \leftarrow |X|\}$
3: **else**
4: randomly select a normal vector $\vec{n} \in IR^{|X|}$ by drawing each coordinate of $\vec{n}$ from a standard Gaussian distribution.
5: randomly select an intercept point $\vec{p} \in IR^{|X|}$ in the range of X
6: set coordinates of $\vec{n}$ to zero according to extension level
7: $X_l \leftarrow filter(X, (X - \vec{p}) \cdot \vec{n} \leq 0)$
8: $X_r \leftarrow filter(X, (X - \vec{p}) \cdot \vec{n} > 0)$
9: **return** inNode$\{Left \leftarrow iTree(X_l, e + 1, l)$,
$Right \leftarrow iTree(X_r, e + 1, l)$,
$Normal \leftarrow \vec{n}$,
$Intercept \leftarrow \vec{p}\}$
10: **end if**

3.2.2 Evaluation stage

The evaluation stage is adjusted correspondingly to the training stage. Focus on line 6 and 8 in Algorithm 6, where the branching is adjusted. As in IF, the average $h(x)$ is computed and provided to the anomaly score formula.

Algorithm 6 $PathLength(\vec{x}, T, e)$

Require: $\vec{x}$ - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
Ensure: path length of $\vec{x}$
1: **if** T is an external node **then**
2: **return** $e + c(T.size)\{c(.)$ is defined in Equation (2.1)$\}$
3: **end if**
4: $\vec{n} \leftarrow T.Normal$
5: $\vec{p} \leftarrow T.Intercept$
6: **if** $(\vec{x} - \vec{p}) \cdot \vec{n} \leq 0$ **then**
7: **return** $PathLength(\vec{x}, T.left, e + 1)$
8: **else if** $(\vec{x} - \vec{p}) \cdot \vec{n} > 0$ **then**
9: **return** $PathLength(\vec{x}, T.rigth, e + 1)$
10: **end if**

3.3 Comparison with Isolation Forest

EIF is a generalization of IF. EIF can fully replace the IF since it allows for leveraging generalization by employing the *extensionLevel* hyperparameter. The minimum value of the hyperparameter is 0, which corresponds to IF behavior. The maximum is $P - 1$ and stands for a fully-extended EIF.

The reason why the IF should still be taken to account as an alternative is the lack of EIF implementations. According to the best knowledge of the data mining platforms, the only current public implementation is the one by authors of [16]. It is a Python implementation that is not supported by any ML platform, which makes it unsuitable for production-ready solutions.

Even if enough implementations of EIF would exist, the IF should remain in use due to its interpretability. Since the IF uses BST inside, then it is easy to interpret why some point was considered an anomaly. Of course, the ensemble of trees makes it difficult also, but there is a chance at least. The interpretability of EIF is irretrievably lost with the adjustment of branching.

3.4 Current implementation of the Extended Isolation Forest

As was said in the previous section, there is a lack of EIF implementations. Only implementation is by the founders [21]. Their implementation will be used as a reference for the one presented in the following sections. The main reasons why their implementation cannot be used in a production environment are:

- Lack of support.
- At this time, the Python package is not working on all platforms according to the opened issue [22].
- Missing seed parameter to make the algorithm deterministic across multiple runs.
- Despite the algorithm itself having low resource requirements, the implementation in Python does not allow for processing of big data.

All these limitations will be corrected in the following chapter by implementing EIF into the H2O-3 library.

Chapter 4

Implementation

There are several options how to make a Machine Learning algorithm public. The quick and easy way is to implement a Python package and publish it on PyPi [23], a repository of all public Python packages. For a user, the R or Python is a well known Machine Learning tool. Therefore by implementing the function in a Python package a wide community of users is covered. Another option is to create a public repository on GitHub [24] for communication with users and/or cooperation with other contributors.

In cases where a Python implementation is insufficient (e,g, for performance reasons) Java can be used instead. Following the implementation the algorithm can be published in the Maven repository [25]. From the user point of view, Java itself is not generally known for ML, and the publishing will take longer because the implementation needs to undergo an approval process.

The best option to make an algorithm usable and user friendly is to contribute the implementation into an open-source platform. Some of these platforms even offer paid user support, which improves the rate at which users' issues or bugs are fixed. This in turn leads to an even better implementation over time.

One such platform is the Python-based Scikit-learn library. However the platform is selective when it comes to adding new algorithms because of the limited resources for code maintenance [26]. In [27] is a list of other Machine Learning platforms with wideranging focuses. Some of them focus on GPU computing [28], scalable Neural Networks [29], scalable linear algebraic operations [30] and more. A Python interface is a *"must have"* requirement for all platforms that intend to be widely used. However, the key for a developer is the computing engine, sometimes it is written in Python itself, but C++, Java, or Scala are also possible alternatives.

Although start contributing to an open-source platform is quite easy, implementing a new algorithm and receiving approval is often a lengthy process, no matter the size or complexity of the implementation. The thesis is a result of cooperation with the H2O.AI company, which agreed to implement Extended Isolation Forest into their H2O-3 library as a complementary offer to the original Isolation Forest that is already implemented.

4.1 H2O.AI

H2O.AI is an open-source software company that offers a variety of products (platforms) and services for Machine Learning. The products help users to build a model and run it on a production system. H2O.AI's focus is providing scalable Machine Learning solutions. The products are designed and prepared not only to process small experimental data sets in a single file, but the same products can also operate with big data technologies such as Hadoop, Spark, and process terabytes of data without user intervention. After training is done, the model can be serialized into a production-ready artifact. In the Figure 4.1 is a portfolio of H2O.AI products. There are both open-source as well as enterprise products in the portfolio, all of them with a possibility of paid support. [31][32]

One of H2O.AI's tools is AutoML technology. It is a tool that helps data scientists with their tasks such as feature extraction, running through ML algorithms and their hyperparameters, finding the best model, and last but not least, deploying the model directly into a production environment. [33][34]

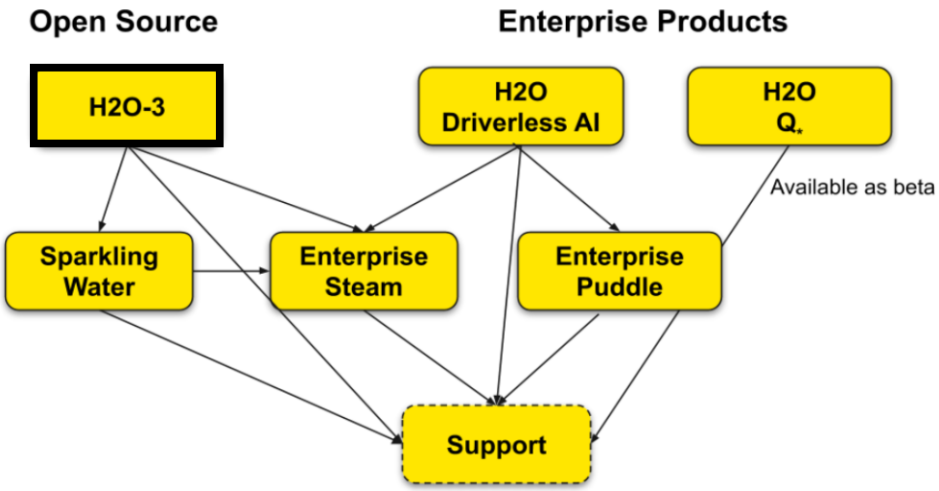


Figure 4.1: H2O.AI technology overview. [35]

4.2 H2O-3 Machine Learning platform

> *"Most simply, H2O-3 is highly scalable, distributed, in-memory, very fast open-source software technology for building AI and Machine Learning (ML) systems."* [35]

H2O-3 is a computing engine written and run as a Java application. It can be run on-premises or in the cloud. H2O-3 has an interface to Python, R, JavaScript, and several more. Users write code in their preferred language, the interface delegates creation of the model to the computing engine and the desired model is returned back. The architecture is shown in Figure 4.2, and the example of H2O-3 Python API is in Listing 4.1, where data is imported from a CSV file and the model is trained. [36]

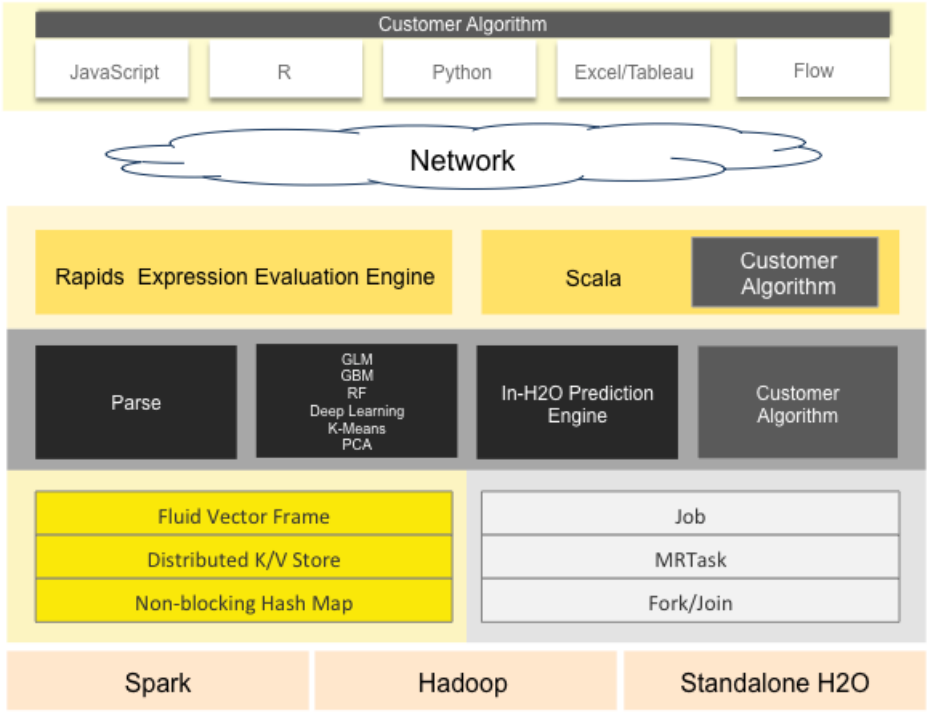


Figure 4.2: H2O-3 Architecture [36]

```
import h2o
from h2o.estimators import
    H2OExtendedIsolationForestEstimator

h2o_df = h2o.importFile("hdfs://path/to/data.csv")
eif = H2OExtendedIsolationForestEstimator(
    model_id = "eif.hex",
    ntrees = 100,
    extension_level = 1)
eif.train(training_frame = hf)
eif_result = eif.predict(h2o_df)
eif_result["anomaly_score"]
...
```

Listing 4.1: Example of model creation with Python in H2O-3

The input file in the Listing 4.1 is imported from the Hadoop file system and can be large. The platform needs to save it in the cluster memory for computing. Usually the table is a 2-D array, but in this case, Java does not allow an array of big data. To resolve this the Distributed Key-Value Store (DKV) is used inside of H2O-3 data structure called Frame [36]. Practically, Frame is a table; it means rows and columns. In H2O-3 terminology, columns are Vectors, and row is an item of Vector. The Frame can be passed to the Map/Reduce task. In such case the given Frame is separated into Chunks (array of chunks). Size of the Chunk array is the same as the number of Vectors. In the Chunks, parts of the rows are equally distributed around the

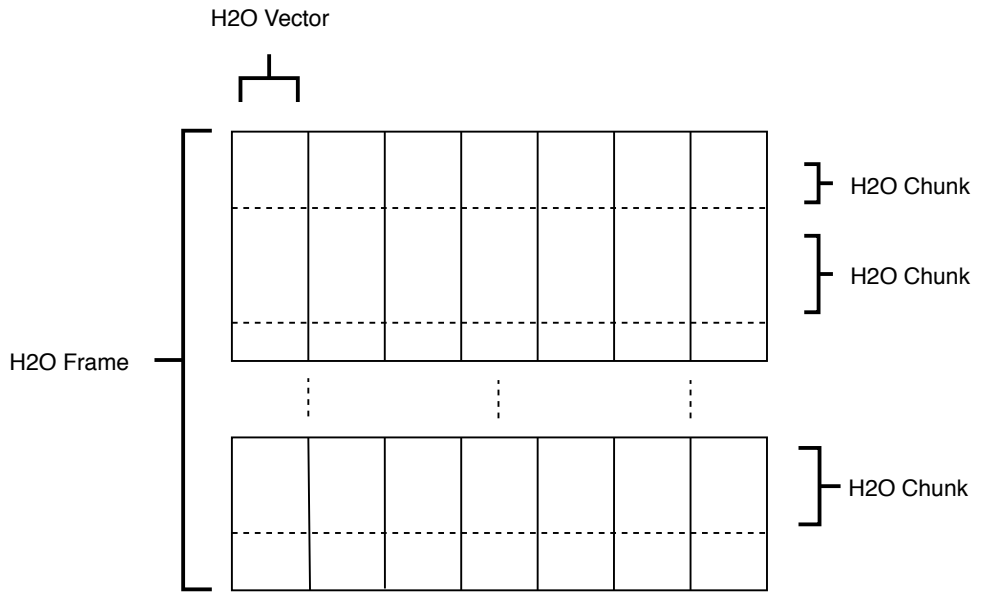


Figure 4.3: Frame

cluster. In this phase the algorithms can change the data. In Figure 4.3 is a diagram of the Frame.

H2O-3 platform also provides an API for the Map/Reduce framework, a Java framework that allows programs to run on a distributed cluster. It is designed to deal with connection and hardware issues without effecting the completion of the calculation. The computing time is increased, but compared to traditional OpenMP/MPI, the problems are handled automatically without a programmer's cooperation. In essence the core of the Map/Reduce framework consists of two operations.

1. Map: distribute a part of the data and the corresponding operation that is performed on this data to the nodes.

2. Reduce: reduce the computed values, e.g. add them and provide a single algorithm output.

H2O.AI uses API for Map/Reduce framework to easily work with data that is in a tabular format . Listing 4.2 shows part of the code responsible for Matrix-Vector multiplication.

```
1  /**
2  * Task for Matrix-Vector product.
3  * Vector is given as last column in task input.
4  */
5  class ProductMtv extends MRTask<ProductMtv> {
6
7  private double[] result;
8
9      /**
10      * Map method. Partial sums are computed.
11      *
12      * @param cs array of chunks.
13      *           Size of array is equal tu number of
           Vectors.
14      *           Last chunk is Vector
15      */
16     @Override
17     public void map(Chunk[] cs) {
18       final int columnLen = cs.length - 1;
19       final Chunk vector = cs[column];
20       result = new double[columnLen];
21       for (int j = 0; j < columnLen; ++j) {
22         double sum = 0.0;
23         for (int i = 0; i < cs[0].len; i++) {
24           sum += cs[j].atd(i) * vector.atd(i);
25         }
26         result[j] = sum;
27       }
28     }
29
30     /**
31      * Reduce method. Partial sums are added together
32      *
33      * @param mrt any finished task from any other worker
34      *            both result arrays contains partial
35      *            sums for each row.
36      */
37     @Override
38     public void reduce(ProductMtv mrt) {
39       ArrayUtils.add(result, mrt.result);
40     }
41 }
```

Listing 4.2: Map/Reduce task example

4.3 Study of the Original Isolation Forest Code

As was already said, the IF is already implemented in the H2O-3 library [37]. Because the EIF algorithm only adjust IF's branching and evaluation it would theoretically be possible to adjust the existing IF implementation. This way the existing API and performance test could be fully taken over. The EIF improves the anomaly score and the paper [16] uses color maps to visualize this. Therefore the same color maps were created. The Scikit-learn implementation of IF and paper implementation of EIF with $extensionLevel = 0$ was visualized together with H2O IF implementation. The Jupiter notebook with the result can be found on the enclosed CD. The input was two blobs of 2-D Gaussian $\mathcal{N}((0,0),(1,1))$ and $\mathcal{N}(10,0),(1,1)$. Figure 4.4 visualizes the anomaly score color map outputted by each algorithm.

It is obvious that the H2O-3 color map in Figure 4.4 does not provide accurate detection of the given data structure. The reason might be missing estimation of possible unsuccessful searches in BST from Formula (2.1). The core of H2O-3 IF is a distributed implementation of BST implemented for the Random Forest algorithm, which does not work with the number of rows

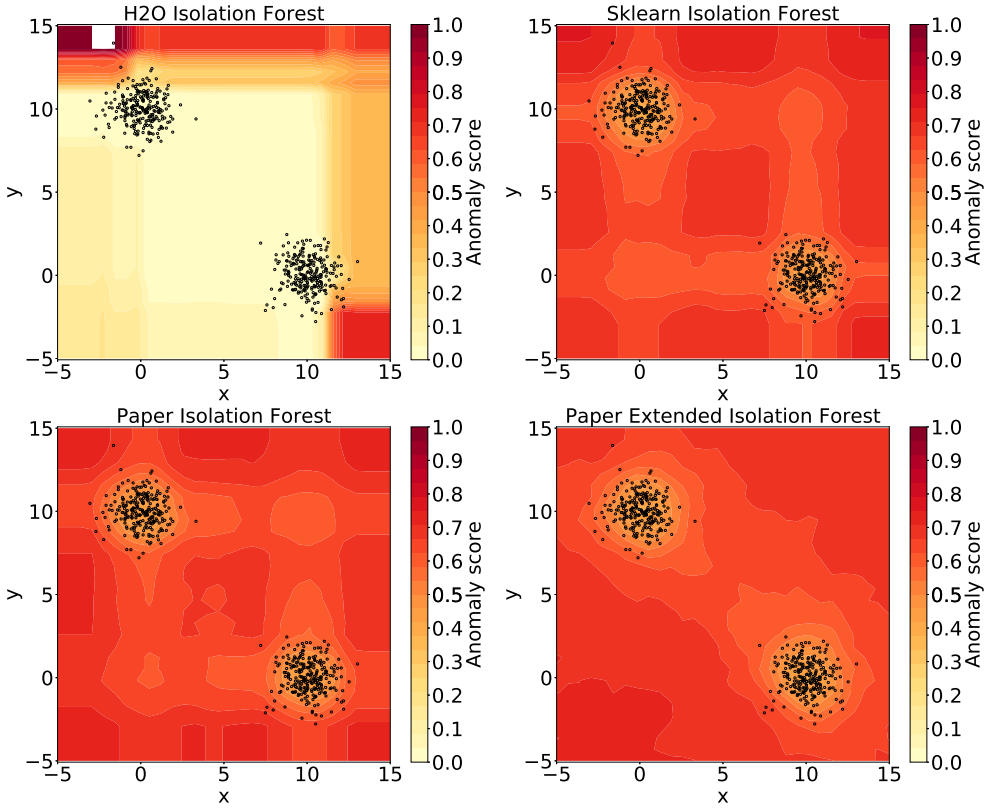


Figure 4.4: Isolation Forest Study

in leaves. Taking into account the detection performance of current Isolation Forest, the fact that the extended version loses interpretability, and last but not least, capabilities, time, and platform knowledge of the author, it was decided that the Extended Isolation Forest would be implemented as a brand new model into H2O-3 platform.

4.4 Parallelization possibilities

The parallelization possibilities for both algorithms and both stages are the same. This section defines the current capabilities of the implemented Isolation Forest and discusses the algorithm's potential. The ability to process big data cannot be affected in any scenario because the H2O-3 data structures are used.

4.4.1 Training Stage

The training stage process is shown in Figure 4.5 and described below. The data are used in read-only mode. Race condition issues are avoided. Isolation Forest builds independent trees with a random sub-sample of data. Each tree can be built separately with reference to a small amount of data. The BST building can also be parallelized. The current Isolation Forest implementation uses a structure called DTree developed for distributed Random Forest algorithms. The DTree structure is responsible for computing performance in current Isolation Forest implementation.

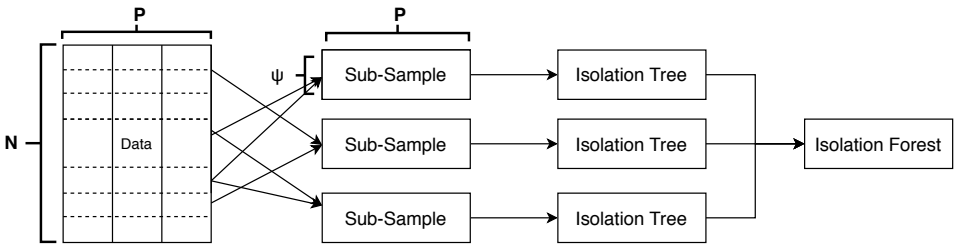


Figure 4.5: Training Stage Diagram

4.4.2 Evaluation Stage

Input for the Forest are the rows of data. The row $h(x)$ is computed in each tree and the average $h(x)$ is the input for the anomaly score formula (2.2). Figure 4.6 graphically depicts the evaluation stage. Path length in each tree can be computed independently. Map/Reduce can be used for this. The Isolation Tree can compute the $h(x)$ in the Map method, and the Reduce method adds the $h(x)$ in the correct output row. Second Map/Reduce task computes the average and anomaly score for each row in the output simultaneously. Cur-

rent Isolation Forest implementation distributes only the input data and path length counting in each tree. The ensemble of trees is sequentially iterated.

The software engineering approach *Simple and Clean Code vs. Performance*, also called *Code clean, performance after* [38][39][40], will be followed during the implementation of the Extended Isolation Forest. A clean and readable code will be implemented first, followed by implementation of all APIs required for the model and last but not least, the model will be tested for anomaly detection and for computing performance. As H2O-3's tools are built for high scalability, further parallelization will have to be implemented. Scalability for big data input/output is possible as long as the correct data structures are used. If the length of an array depends on the number of rows, then the Map/Reduce task must be implemented. If the length depends on a number of features, then simple array operations are used. Nevertheless, the first implementation will be based on sequential Isolation Tree structure and sequential evaluation stage.

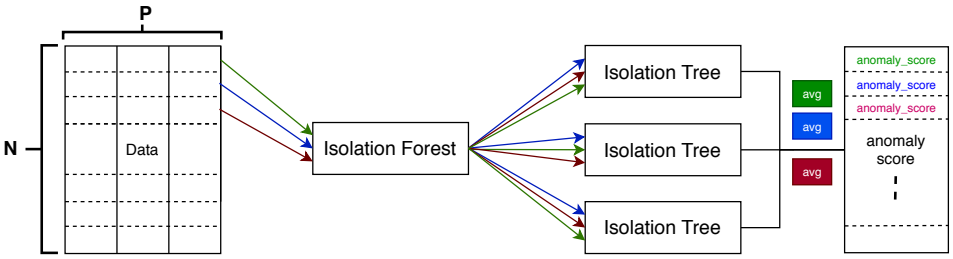


Figure 4.6: Evaluation Stage Diagram

4.5 Model Implementation

The task of implementing a model into the existing H2O-3 platform does not differ from other usual software developer tasks. The status of EIF implementation is tracked in the issue ticket task PUBDEV-7138 [41]. The task included standard activities, such as understanding the code base, understanding the interface, following the given code convention, feature merging process and many more. According to the work log the implementation took approximately 200MH - a short section below describes the main activities that have been done.

The Extended Isolation Forest model has following hyperparameters:

`ntrees` = number of trees in ensemble.

`sample_size` = size of sub-sample

`extension_level` = level of extension

The implemented Java/Python classes and R files are listed below together with a short description. All the implemented files are provided on the enclosed CD and they are also available in the Pull Request [42].

- `ExtendedIsolationForest.java`

Contains the training stage of EIF.

- `ExtendedIsolationForestTest.java`

Contains automatic tests for EIF, including basic tests for the training stage as well as helper methods.

- `ExtendedIsolationForestModel.java`

Contains the evaluation stage of EIF and holds the trained trees.

- `ExtendedIsolationForestMojoWriter.java`

Classes used to save the model in the cluster.

- `FilterGteTask.java` and `FilterLtTask.java`

Map/Reduce tasks for filtering the data by given $\vec{n}$ and $\vec{p}$. See Algorithm 5 line 7 and 8 for more information.

- `SubSampleTask.java`

Map/Reduce task for sub-sampling of the given data.

- `ExtendedIsolationForest.IsolationTree`

Whole implementation of Algorithm 5. Recursion is replaced with an array implementation of BST, which is more suitable for use in distributed computing.

- `gen_{alogrithm}.py` (e.g. `gen_python.py` and `gen_R.py`)

The reflection ability is used to generate APIs for various programming languages. Individual files are the API generation codes for each language.

- `extended_isolation_forest.py` and `extendedisolationforest.R`

Automatically generated Python and R files for the Extended Isolation Forest algorithm.

4.6 Model Testing

Testing can be divided into three processes. First is Unit testing of the implemented class and of the helper methods. Unit testing has been carried out as it is a project policy to provide automatic tests for the new code. The process of unit testing will not be described in this section. Second is testing of anomaly detection performance on toy data. Third is testing of the algorithm scalability.

4.6.1 Anomaly detection performance

The performance is tested on 2-D toy data sets. The paper [16] evaluates the anomaly detection performance by comparing mean and variance of the given anomaly score. The same method is used in this section. Eight data sets were generated, with each data set subsequently trained on ten models of H2O EIF and one model by the paper implementation of EIF (because of computing time). The results are in Figure 4.10. The means of both algorithms are similar, but the variance for H2O EIF is about 0.004 higher than the paper implementation. According to the results in [16] it should mean that the paper implementation provides a more accurate detection of the structure of the given data. This is especially true on data sets 6 (Figure 4.8) and 7 (Figure 4.9). The reason for the higher variance in H2O model could be insufficient robustness of the sub-sampling method of H2O's implementation. It is possible that the method suffers from bias and prefers the point that is selected as first in the addition test. The enclosed CD contains the Jupyter notebook with other tested data sets.

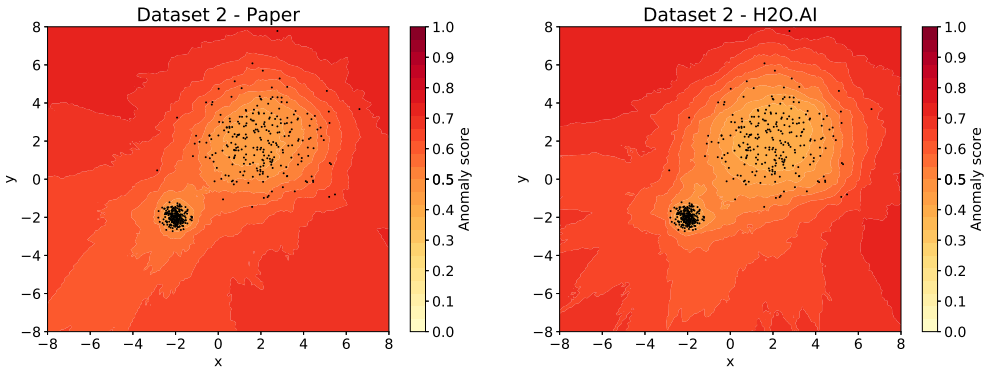


Figure 4.7: Anomaly score map - dense and sparse blob

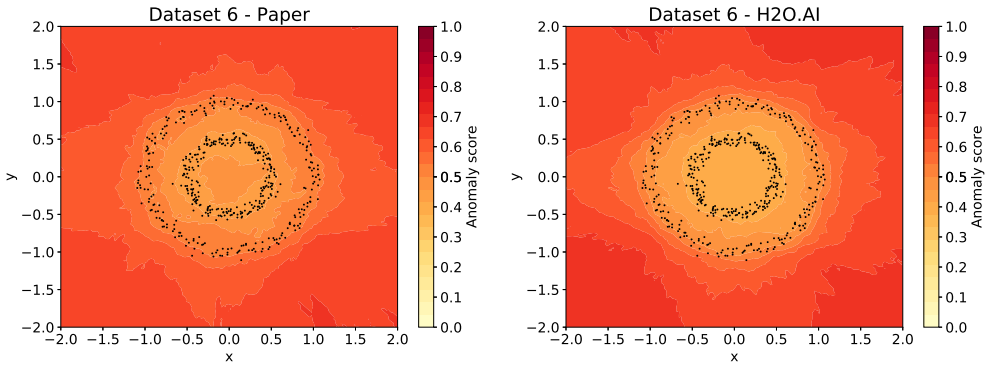


Figure 4.8: Anomaly score map - data circle

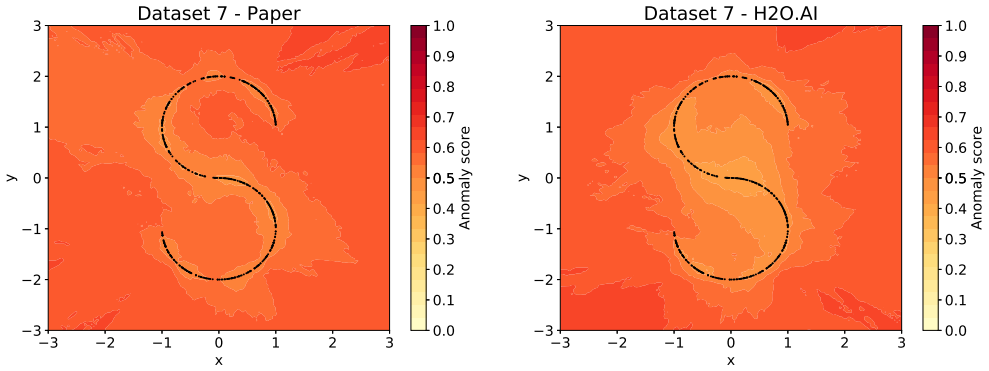


Figure 4.9: Anomaly score map - data S

4.6.2 Scalability

Scalability of the training and evaluation stages are tested separately. Scalability performance is compared to H2O-3's implementation of IF. The tests are done on laptop hardware:

- Lenovo ThinkPad P53,
- MS Windows 10 Pro x64,
- Intel Core i7-9850H CPU @ 2.60GHz,
- 6 cores and 12 threads,
- 96.0 GB RAM.

Ten models are trained with the same settings:

- ntrees = 100,
- seed = 1234.

The Jupyter notebook is available on the enclosed CD.

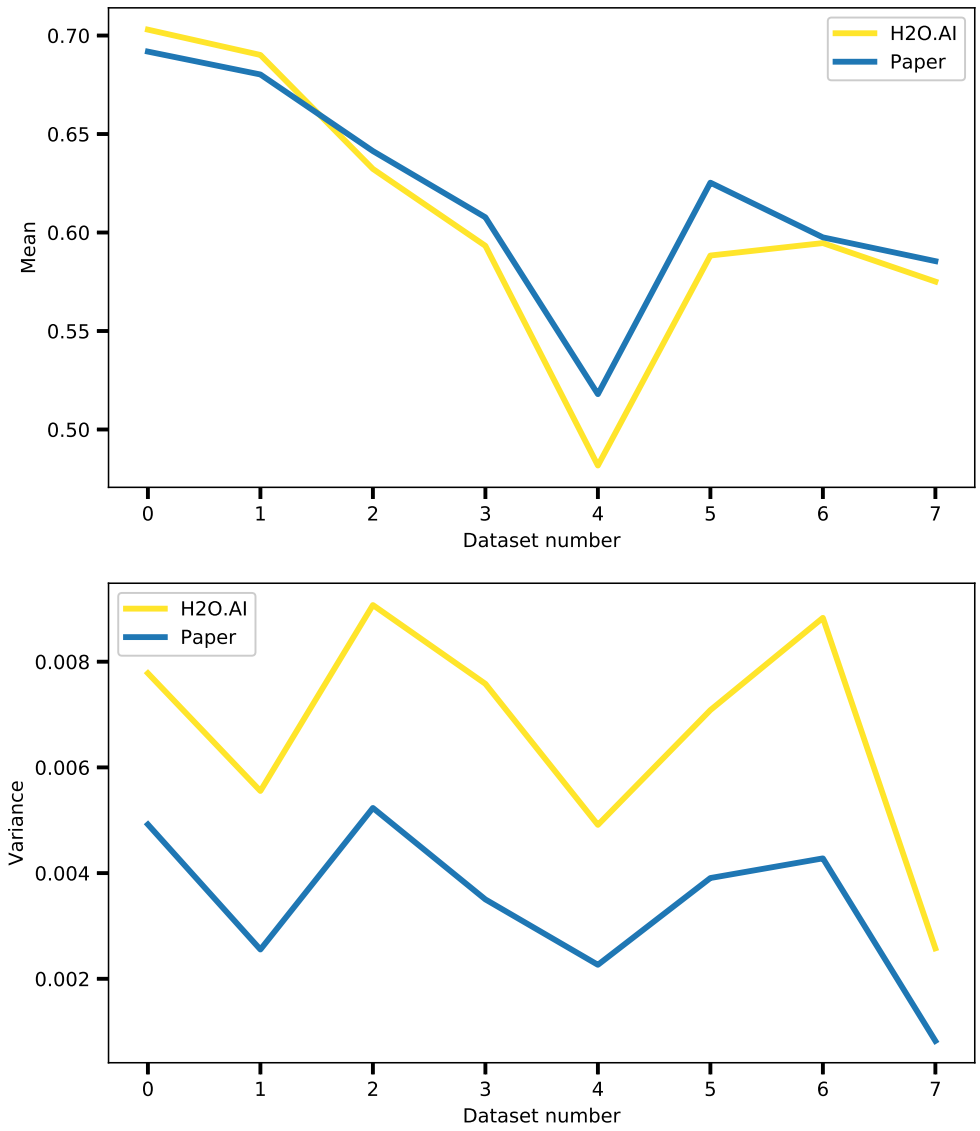


Figure 4.10: Anomaly detection performance

4.6.2.1 Training stage

Since parallelization is only relevant for big data input, the scalability depends on the *sample_size* parameter. For small values, e.g. the recommended 256, the computing is sequential except for sub-sampling. For the *sample_size* values where Map/Reduce helps, the performance significantly depends on the number of threads. For a large *sample_size* DTree works well and it was worth implementing BST as a distributed version. If the *sample_size* is small then the DTree slows down the computing. In Figure 4.11 Extended Isolation

Forest performs better on large data when *sample_size* is kept small. In this case computing time is lower even with fewer threads. Isolation Forest performs worse with a small *sample_size* most probably because of the DTree structure. On the other hand, IF performs significantly better with a large *sample_size*. The number of $iTree$ nodes in EIF depends on *sample_size* defined in Equation (4.1). In IF the same function is provided by the max_depth attribute.

$$\begin{aligned} height &= ceiling(log_2(sample\_size)) \\ \#nodes &= 2^{height} - 1 \end{aligned} \tag{4.1}$$

Suppose data with 1 500 000 rows and $sample\_size = 15\ 000$, which is 1% of the data. It means $\#nodes \approx 2^{14} - 1 = 16\ 383$. In this case, DTree or any distributed BST is suitable as Figure 4.12 shows. The reason why computing time increase for Extended Isolation Forest, is the required communication between threads when data is distributed but the computing is running in only one thread.

4.6.2.2 Evaluation stage

The evaluation stage performs similarly for both implementations. The logarithmic dependence on number of threads is clearly visible in Figure 4.13 and Figure 4.14. In this case it does not matter if the $iTree$ is distributed, nor does it matter what value the *sample_size* has.

4.6.2.3 Scalability Test Outcome

The evaluation stage has good performance for both algorithms. Computing time is logarithmic smaller with the number of threads. There are options to be more scalable in section 4.4.2, but from the software engineering point of view, there is no significant added value that would justify the effort. The added value lies in the training stage. Since the *sample_size* is small, the sequential algorithm performs better as Figure 4.11 shows. When *sample_size* increases, the IF computing time is logarithmically smaller with the number of threads, but the computing time of EIF is increasing with the number of threads because of the communication between threads with no added value in computing time. To fix this EIF issue, both scalability options presented in section 4.4.2 are equally acceptable for model users. If the data is large and *sample_size* kept small, then the number of trees will most probably increase, and in that case, each tree can be built independently with the sequential algorithm. If the *sample_size* increases, then the number of trees is most likely kept small and distributed BST helps more than the sequential algorithm.

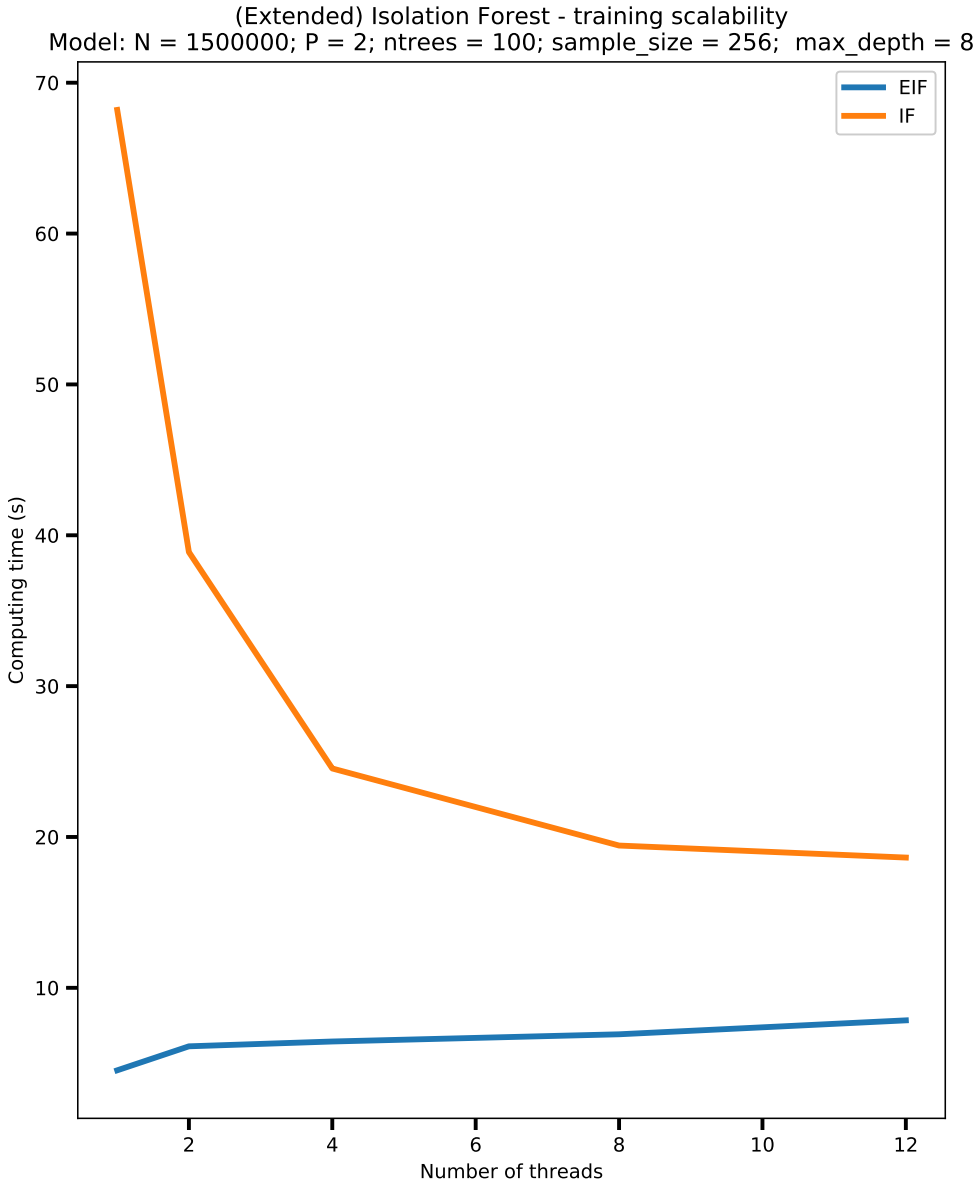


Figure 4.11: Scalability - training stage with $sample\_size = 256$

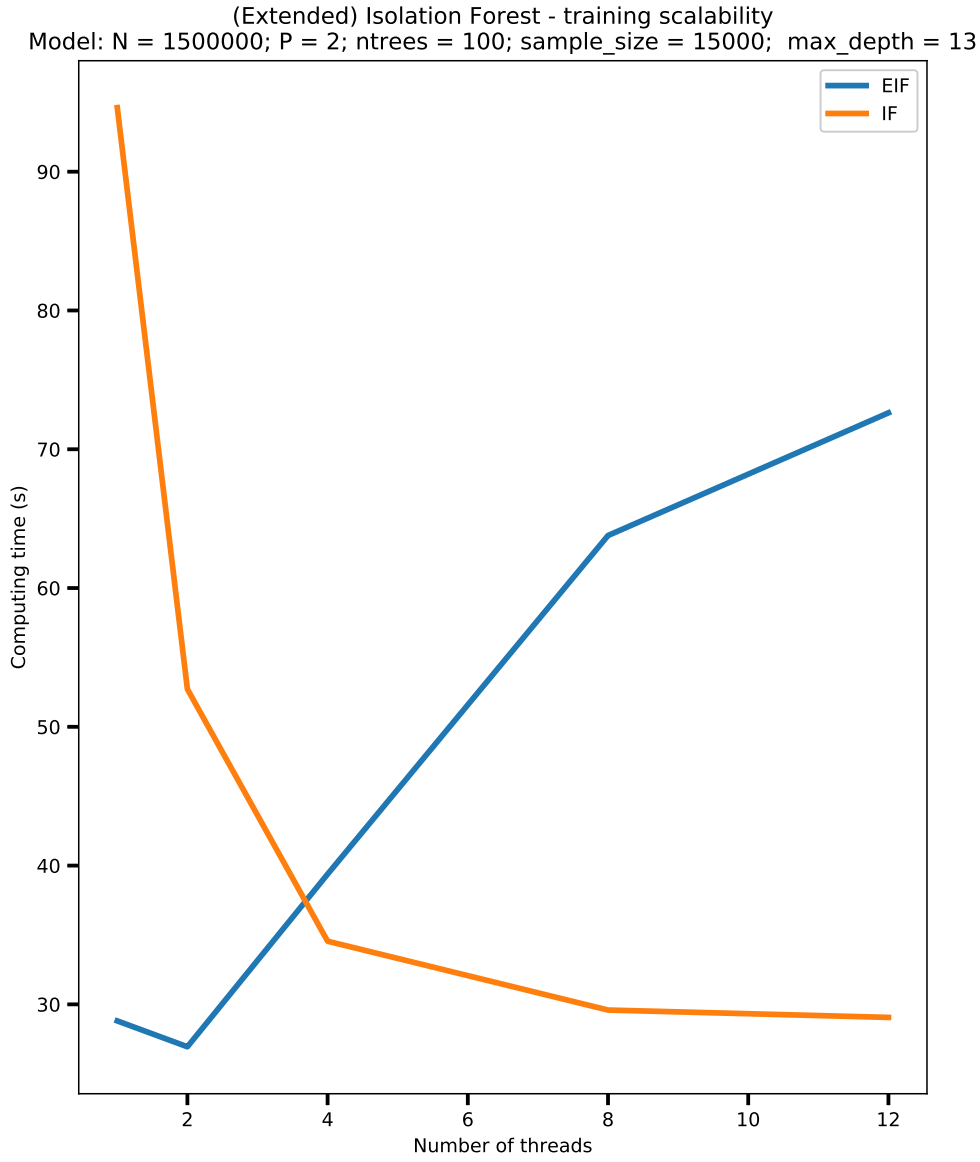


Figure 4.12: Scalability - training stage with $sample\_size = 15\ 000$

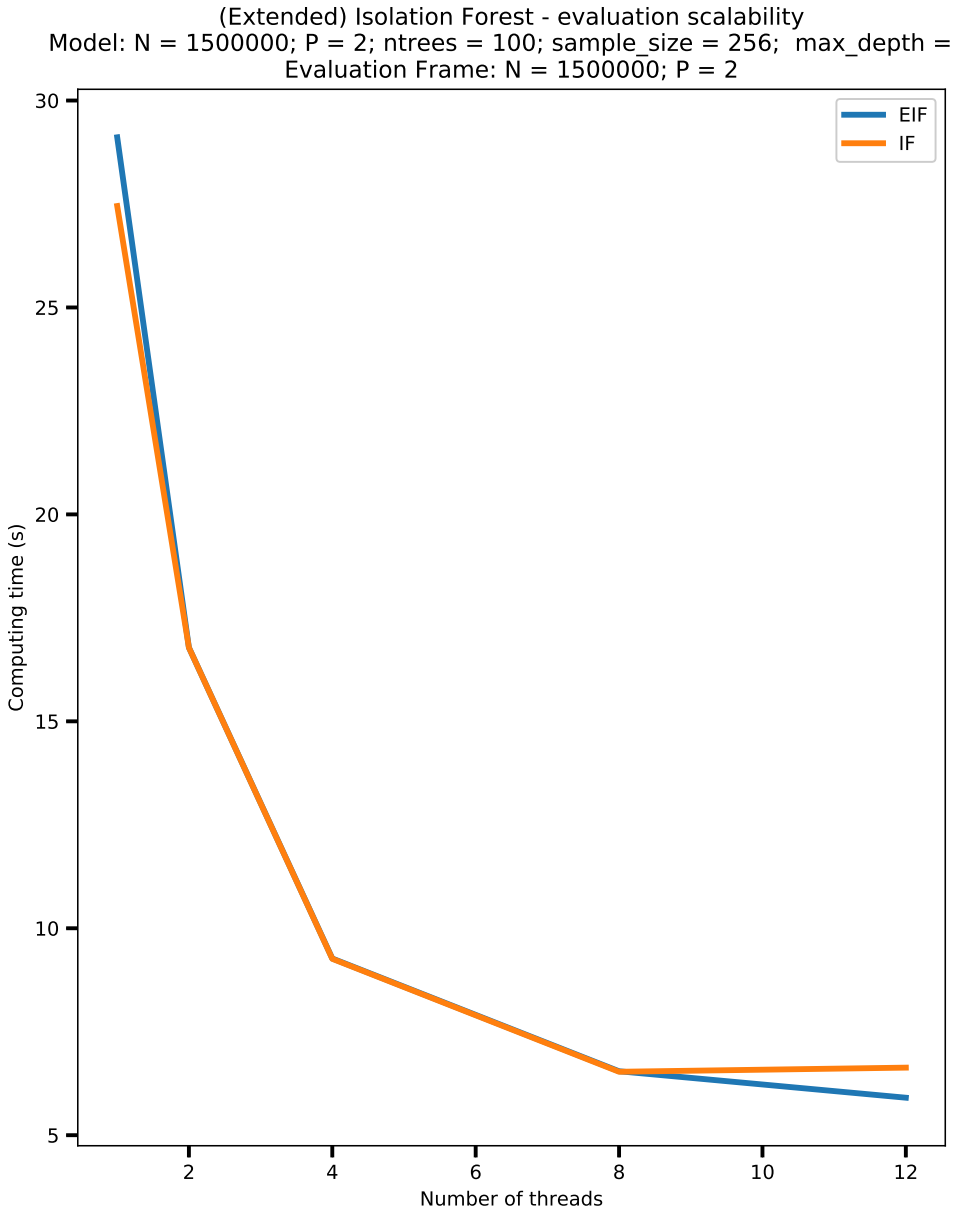


Figure 4.13: Scalability - evaluation stage with $sample\_size = 256$

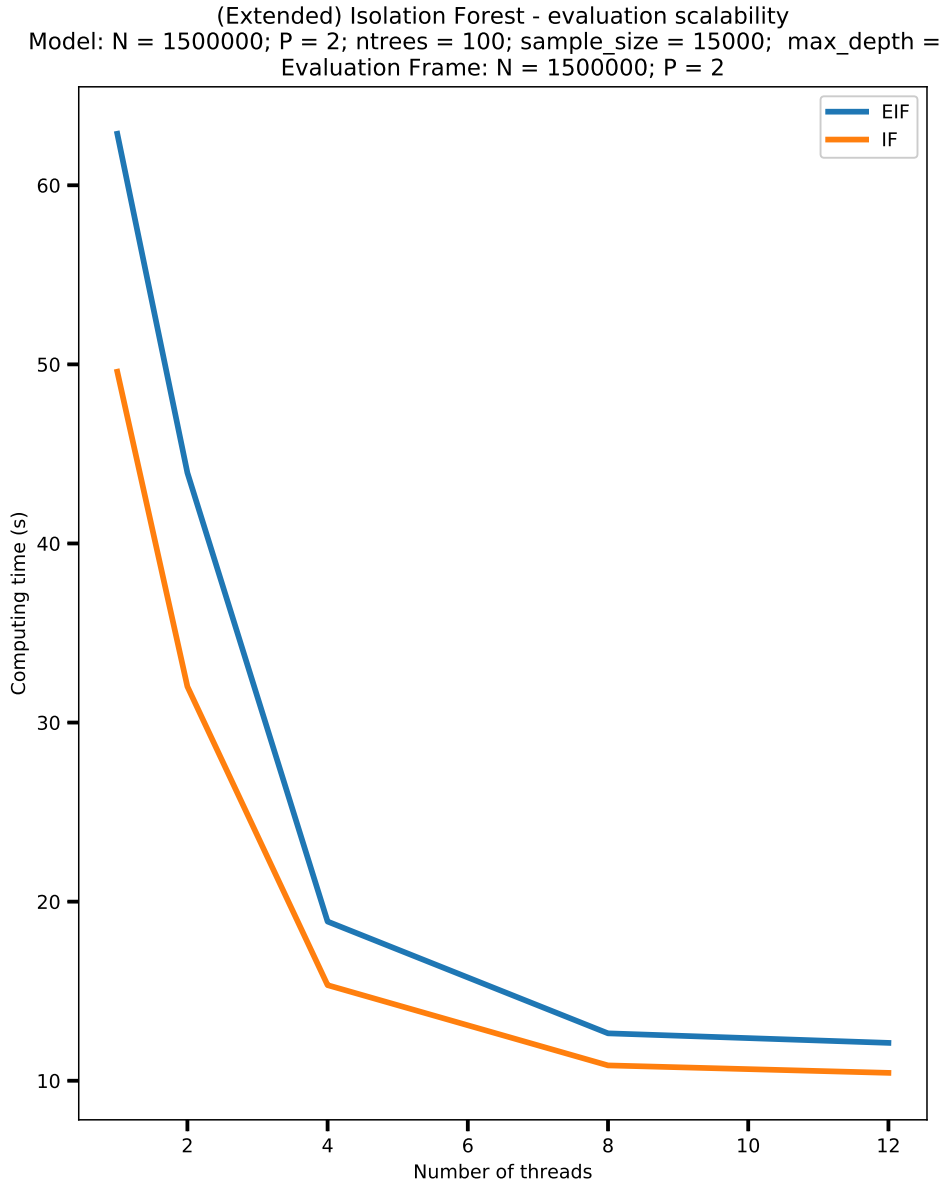


Figure 4.14: Scalability - evaluation stage with $sample\_size = 15\ 000$

Conclusion

The thesis dealt with the implementation of the Extended Isolation Forest algorithm into the H2O-3 Machine Learning open-source platform. In the first part, the Anomaly Detection algorithms were introduced - both from the field of Machine Learning (K-means, DBSCAN, SVM) as well as from the Deep Learning branch (Robust Deep Autoencoder).

The second part presented the idea of Isolation Forest, the rationale of Extended Isolation Forest, and continued with the third part, Extended Isolation Forest implementation. It was decided to implement a brand new model and leave the currently implemented Isolation Forest model untouched. The implementation was successful and the current status is Pull Request waiting for review. The test results showed that the Extended Isolation Forest Model needs to be adjusted. Anomaly detection performance tests revealed a slight imperfection in the detection of the data structure when compared to the only available Python implementation of the algorithm. The issue could be fixed with a better sub-sampling or in the $iTree$ branching, where the algorithm could avoid split points with an empty ancestor.

Scalability tests were done and results compared to the current H2O-3 Isolation Forest implementation. The tests for evaluation stage passed. The computing time is logarithmically smaller with the number of threads. On the other hand, a performance gap was found in the Training stage. For the Extended Isolation Forest, the computing time does not change when the $sample\_size$ value is kept small. However, with a larger $sample\_size$, the computing time requirements for Extended Isolation Forest increase, whereas computing time for Isolation Forest gets logarithmically smaller with the number of threads in all cases. The training performance can be increased by building the trees in parallel. It could even outperform the current Isolation Forest. Another possibility to improve performance is to do the tree branching process in parallel.

I will continue to develop the Extended Isolation Forest implementation in close cooperation with H2O - not only to resolve the mentioned issues but to further improve it. The status of the production-ready deployment is in the issue tracking system ticket. As mentioned in the Extended Isolation Forest paper, the algorithm could be further improved by scaling the anomaly detection for high-dimension data. This could be implemented by adding another parameter that would allow for a feature selection method in the computation.

Bibliography

[1] Chandola, V.; Banerjee, A.; et al. Anomaly Detection: A Survey. *ACM Comput. Surv.*, volume 41, 07 2009, doi:10.1145/1541880.1541882.

[2] Zimek, A.; Filzmoser, P. There and back again: Outlier detection between statistical reasoning and data mining algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, volume 8, no. 6, Nov. 2018, ISSN 1942-4787, doi:10.1002/widm.1280.

[3] Chalapathy, R.; Chawla, S. Deep Learning for Anomaly Detection: A Survey. *CoRR*, volume abs/1901.03407, 2019, 1901.03407. Available from: `http://arxiv.org/abs/1901.03407`

[4] Liu, F. T.; Ting, K.; et al. Isolation Forest. 01 2009, doi:10.1109/ICDM.2008.17.

[5] Scikit-learn. *IsolationForest*. Scikit-learn, [cit. 2020-04-17]. Available from: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html`

[6] Adnan, R.; Mohamad, M. N.; et al. Multiple Outliers Detection Procedures in Linear Regression. *Matematika*, volume 19, 01 2003.

[7] Mishra, A. *Swamping and Masking in Anomaly detection: How Subsampling in Isolation Forests helps mitigate this?* Medium, [cit. 2020-04-17]. Available from: `https://medium.com/walmartlabs/swamping-and-masking-in-anomaly-detection-how-subsampling-in-isolation-forests-helps-mitigate-bb192a8f8dd5`

[8] Münz, G.; Li, S.; et al. Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, 2007, pp. 13–14.

[9] DataScience.com. *Introduction to Anomaly Detection*. KDnuggets, [cit. 2020-04-17]. Available from: `https://www.kdnuggets.com/2017/04/datascience-introduction-anomaly-detection.html`

[10] Ahmed, M.; Mahmood, A. N.; et al. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, volume 60, 2016.

[11] Gumbao, M. G. *Best clustering algorithms for anomaly detection.* Towards Data Science, [cit. 2020-04-17]. Available from: https://towardsdatascience.com/best-clustering-algorithms-for-anomaly-detection-d5b7412537c8

[12] Schubert, E.; Sander, J.; et al. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.*, volume 42, no. 3, July 2017, ISSN 0362-5915, doi:10.1145/3068335. Available from: https://doi.org/10.1145/3068335

[13] Ester, M.; Kriegel, H.-P.; et al. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 1996, pp. 226–231.

[14] Li, X.; Lu, Y.; et al. Out-of-Distribution Detection for Skin Lesion Images with Deep Isolation Forest. 2020, 2003.09365.

[15] Ding, Z.; Fei, M. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window. *IFAC Proceedings Volumes*, volume 46, no. 20, 2013: pp. 12 – 17, ISSN 1474-6670, doi:https://doi.org/10.3182/20130902-3-CN-3020.00044, 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013. Available from: http://www.sciencedirect.com/science/article/pii/S1474667016314999

[16] Hariri, S.; Carrasco Kind, M.; et al. Extended Isolation Forest. *IEEE Transactions on Knowledge and Data Engineering*, 2019: p. 1–1, ISSN 2326-3865, doi:10.1109/tkde.2019.2947676. Available from: http://dx.doi.org/10.1109/TKDE.2019.2947676

[17] Lutins, E. *DBSCAN: What is it? When to Use it? How to use it.* Medium, [cit. 2020-04-17]. Available from: https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818

[18] Zhang, X.; Gu, C.; et al. Support Vector Machines for Anomaly Detection. 01 2006, pp. 2594 – 2598, doi:10.1109/WCICA.2006.1712831.

[19] Zhou, C.; Paffenroth, R. C. Anomaly Detection with Robust Deep Autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, New York, NY, USA: Association for Computing Machinery, 2017, ISBN 9781450348874, p. 665–674, doi:10.1145/3097983.3098052. Available from: https://doi.org/10.1145/3097983.3098052

[20] Wikipedia contributors. Euler–Mascheroni constant — Wikipedia, The Free Encyclopedia. 2020, [Online; accessed 8-May-2020]. Available from: https://en.wikipedia.org/w/index.php?title=Euler%E2%80%93Mascheroni_constant&oldid=954141487

[21] Hariri, S.; Carrasco Kind, M.; et al. Extended Isolation Forest. *IEEE Transactions on Knowledge and Data Engineering*, 2019: pp. 1–1, doi:10.1109/TKDE.2019.2947676. Available from: https://github.com/sahandha/eif

[22] PoradaKev. *Error while installing eif*. GitHub, [cit. 2020-04-17]. Available from: https://github.com/sahandha/eif/issues/14

[23] joelbarmettlerUZH. *How to upload your python package to PyPi*. Medium, [cit. 2020-04-17]. Available from: https://medium.com/@joel.barmettler/how-to-upload-your-python-package-to-pypi-65edc5fe9c56

[24] GitHub. *GitHub*. GitHub, [cit. 2020-04-17]. Available from: https://github.com/

[25] Foundation, T. A. S. *Guide to uploading artifacts to the Central Repository*. The Apache Software Foundation, [cit. 2020-04-17]. Available from: https://maven.apache.org/repository/guide-central-repository-upload.html

[26] Scikit-learn. *Contributing*. Scikit-learn, [cit. 2020-04-17]. Available from: https://scikit-learn.org/stable/developers/contributing.html

[27] Roy, S. *13 Open-Source Artificial Intelligence and Machine Learning Tools to Watch in 2019*. DEV, [cit. 2020-04-17]. Available from: https://dev.to/promozseo/13-open-source-artificial-intelligence-and-machine-learning-tools-to-watch-in-2019-1hmc

[28] TensorFlow. *Contribute to TensorFlow*. TensorFlow, [cit. 2020-04-17]. Available from: https://www.tensorflow.org/community/contribute

[29] Caffe. *Development and Contributing*. Caffe, [cit. 2020-04-17]. Available from: https://caffe.berkeleyvision.org/development.html

[30] MAHOUT. *How to contribute*. MAHOUT, [cit. 2020-04-17]. Available from: https://mahout.apache.org/developers/how-to-contribute

[31] H2O.AI. *H2O.ai is Democratizing Artificial Intelligence.* H2O.AI, [cit. 2020-04-17]. Available from: https://www.h2o.ai/company/

[32] H2O.AI. *H2O.ai Products and Solutions*. H2O.AI, [cit. 2020-04-17]. Available from: https://www.h2o.ai/products/

[33] H2O.AI. *AutoML: Automatic Machine Learning*. H2O.AI, [cit. 2020-04-17]. Available from: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html

[34] Matoušek, J. *AutoML - NAHRADÍ ROBOTI ANALYTIKY?* Data Mind, [cit. 2020-04-21]. Available from: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html

[35] Ellen Friedman, P. *AI & ML Platforms: My Fresh Look at H2O.ai Technology*. H2O.AI, [cit. 2020-04-17]. Available from: https://www.h2o.ai/blog/ai-ml-platforms-my-fresh-look-at-h2o-ai-technology/

[36] H2O.AI. *H2O Architecture*. H2O.AI, [cit. 2020-04-17]. Available from: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/architecture.html

[37] H2O.AI. *Isolation Forest*. H2O.AI, [cit. 2020-05-17]. Available from: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/if.html

[38] simon. *Should a developer aim for readability or performance first? [closed]*. Stack Overflow, [cit. 2020-04-17]. Available from: https://stackoverflow.com/questions/183201/should-a-developer-aim-for-readability-or-performance-first

[39] akmad. *Performance vs Readability*. Stack Overflow, [cit. 2020-04-17]. Available from: https://stackoverflow.com/questions/30754/performance-vs-readability

[40] Mertz, A. *Simple and Clean Code vs. Performance*. Simplify C++!, [cit. 2020-04-17]. Available from: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/architecture.html

[41] Maurerová, V. *Implement Extended Isolation Forest*. H2O.AI, [cit. 2020-05-25]. Available from: https://0xdata.atlassian.net/browse/PUBDEV-7138

[42] Valenta, A. *PUBDEV-7138 Extended Isolation Forest*. H2O.AI, [cit. 2020-05-25]. Available from: https://github.com/h2oai/h2o-3/pull/4319

Appendix A

Acronyms

AI Artificial Intelligence. 31

BDT Binary Decision Tree. 15

BST Binary Search Tree. 17, 19, 20, 23, 25, 28, 35, 36, 41, 42

DBSCAN . 9, 11, 21, 47

DeepIF Deep Isolation Fores. 11

DKV Distributed Key-Value Store. 32

DL Deep Learning. 7, 8, 47

DTree Distributed Tree. 36

EIF Extended Isolation Forest. 2, 8, 23, 26, 28, 30, 35, 37–39, 42, 47, 48

GPU Graphics Processing Unit. 29

HDBSCAN* Hierarchical Density-Based Spatial Clustering. 11

IF Isolation Forest. 2, 8, 13, 15–18, 20, 21, 23, 25–28, 30, 35, 36, 40, 42, 47

iForestASD Isolation Forest Adapted Streaming Data. 11

iTree Isolation Tree. 15

LSDBC Locally Scaled Density Based Clustering. 11

LSTM Long short-term memory. 12

MH Man Hour. 37

ML Machine Learning. 8, 28–31, 47

NN Neural Networks. 7–9, 12

OPTICS Ordering Points To Identify the Clustering Structure. 11

PCA Principal component analysis. 12

R The R Project for Statistical Computing. 29

RDA Robust Deep Autoencoder. 12, 13

RNN Replicator Neural Networks. 12

SVM Support Vector Machine. 8, 9, 12, 47

Appendix B

Contents of enclosed CD

- `eif-anomaly-perf.ipynb` ..... EIF anomaly detection performance study
- `eif-scalability-train.ipynb` ..... EIF training stage scalability study
- `eif-scalability-eval.ipynb` .... EIF evaluation stage scalability study
- `h2o-3` ..... Implementation sources
  - `h2o-algos` ..... Folder with all algorithms
    - `src/main/java/hex/tree/isofor` ..... IF implementation
    - `src/main/java/hex/tree/isoforextended` .. EIF implementation
  - `h2o-py`
    - `demos/extisofor` ..... Helper notebooks
- `text`
  - `src` ..... Thesis text source code
  - `thesis.pdf` ..... Thesis text in PDF format