Extending Isolation Forest to support non-numerical data

from

Sina Barghidarian

Master's Thesis in Computer Science

submitted to

Faculty of computer science

Technical University of Dortmund

March, 2022

at

Chair of Data Science and Data Engineering

at

Frist supervisor: Prof. Dr. Emmanuel Müller
Second supervisor: Simon Klüttermann

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dortmund, der 29. März, 2022

Contents

Abstract

In machine learning, the presence of rare and unusual data points in training datasets has led to the concept of anomaly detection. The majority of anomaly detection algorithms are only compatible with numerical attributes and as a result, the categorical attributes will often be ignored in practice. Although there are approaches to overcome this issue, for example by converting the categorical attributes to numerical ones using one-hot encoding, it is more efficient to have algorithms that handle such attributes inherently. Isolation Forest is an unsupervised anomaly detection algorithm, which in its original version supports only numerical data. In this work we propose a method to support categorical attributes and to some extent textual data for Isolation Forest. This work is in collaboration with *Continentale Krankenversicherung a.G.* who wanted to use a machine learning technique to detect harmful requests, i.e. anomalies, in HTTP traffic logged by a *Web Application Firewall (WAF)*. One of the challenging parts of this work was that the majority of the attributes of the WAF's training data were either categorical or textual, which motivated us to extend Isolation Forest to support non-numerical attributes.

In chapter 1, we look at the inner workings of the Isolation Forest and two other anomaly detection techniques, namely Autoencoder and Local Outlier Factor(LOF). In chapter 2 related works on the Isolation Forest regarding the support for categorical features are shown. After that, we present our extension to the Isolation Forest. Based on our idea for categorical data we present in chapter 3 our second extension for Isolation Forest, which enables it to support textual data without any pre-processing on text data. In chapter 4 we evaluate the performance of the new Isolation Forest, Autoencoder, and LOF by conducting several experiments using 13 public datasets and one synthetic dataset. We observe that our Isolation Forest performs much better than its competitor in terms of ROC AUC and it also benefits more from the categorical attributes. Finally, in chapter 5 we use Isolation Forest to find new types of malicious behavior on the WAF dataset.

By the end of this work, the advantages of Isolation Forest for categorical data shall be clear to the reader. Not only because of its performance that surpasses other methods like the Autoencoder but also because of the neat properties of this algorithm that has allowed us to extend it in order to support non-numerical features. It is also important to know that the categorical features often contribute to better performance and therefore one should not simply ignore them because an algorithm does not support such features.

1

Anomaly detection Methods

Anomalies are data points that have characteristics other than the normal data points and are usually rare. They are likely to be found in every dataset and therefore the detection of anomalies is an interesting area of study in machine learning. Detecting fraudulent transactions in credit card transactions, rare diseases in patient's medical records or unusual activities in computer network traffic are usecases of anomaly detection. This chapter introduces three popular methods to detect anomalies that are used in later chapters: Isolation Forest, Autoencoder and Local outlier factor (LOF).

1.1 Isolation Forest

Isolation Forest [1] is an unsupervised anomaly detection technique that builds an *ensemble* (meaning a collection) of binary trees, called *Isolation Trees* (or *iTrees*), from a given dataset. It is similar to a Random Forest [2] in the sense that each iTree in the ensemble makes a prediction for a given data point and the final outcome of the algorithm is an aggregation of the individual predictions of the iTrees.

Each iTree partitions the instances of a given dataset randomly in a recursive manner until all the instances are isolated. Since anomalies differ from the rest of the data and have distinguishable attribute-values, they are likely to be isolated with fewer partitions. Therefore they are closer to the root of the tree. In other words anomalies tend to have shorter *path lengths*. Hence, when an ensemble of iTrees result in shorter path lengths for a given data point, then that point is likely to be an outlier. Each iTree is built upon a sub-sampled dataset with a size of ψ by randomly selecting instances from the training dataset without replacement.

1.1.1 Isolation using Isolation Tree

An Isolation Tree is a *proper binary tree* where every internal node has exactly two child nodes (T_l, T_r) and the nodes with no children are called external nodes or leaf nodes. Each internal node encapsulates a test over an attribute q with a split value p, that is, the data points that fulfill the test $q < p$ go to the left child node T_l and the rest go to the right child node T_r.

Let $X = \{x_1, ..., x_n\}$ be a dataset with n instances and d dimensions. An Isolation Tree begins at the root of the tree by randomly selecting an attribute q and a random split point p such that p is between the minimum and maximum values of the attribute q. The instances of the X would be sorted into the left and the right child nodes of the current internal node based on the test criteria $q < p$. This procedure is repeated recursively until either the tree reaches it's height limit or $|X| \leq 1$. The details of the construction of an Isolation Tree during the training stage is demonstrated by algorithm 1.

Algorithm 1: *iTree(X, e, l)*

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

```
1  if e ≥ l or |X| ≤ 1 then
2  |  return exNode{Size ← |X|}
3  else
4  |  let Q be a list of attributes in X
5  |  randomly select an attribute q ∈ Q
6  |  randomly select a split point p from max and min values of attribute q in X
7  |  X_l ← filter(X, q < p)
8  |  X_r ← filter(X, q ≥ p)
9  |  return inNode{Left ← iTree(X_l, e + 1, l), Right ←
   |   iTree(X_r, e + 1, l), SplittAtt ← q, SplitValue ← p}
10 end
```

1.1.2 Building Isolation Forest from Isolation Trees

Training of a single Isolation Tree was explained in the previous section. Having created an Isolation Tree, it is now pretty straight forward to build an ensemble of such trees during the training's process of an Isolation Forest. The details of this process is to be found in algorithm 2.

Algorithm 2: $iForest(X, t, \psi)$

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

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 $\quad Forest \leftarrow Forest \cup iTree(X^{'}, 0, l)$
6 **end**
7 **return** $Forest$

There are only two parameters for an Isolation Forest. The number of Isolation Trees, t, in the forest and the sub sampling size ψ.

The number of trees t specifies the size of the forest. As recommended by the authors of this method $t = 100$ shall be the default value. Because the path lengths converge usually before this value.

Sub sampling size ψ controls the size of the training dataset $X^{'}$ for each single Isolation Tree. When an optimal value for ψ is found, then there is no need to increase that further. Because by doing so the processing time and the memory usage increases without any benefits in detection performance. The authors of Isolation Forest show that with $\psi = 256$ the performance of Isolation Forest is near optimal.

In the second line of the algorithm 2 the height limit for Isolation Trees is set according to the sub sampling size ψ, that is $l = ceil(log_2\ \psi)$. This value is approximately the average height of a binary tree [3]. After that for each Isolation Tree a sub sampled dataset $X^{'}$ is populated by randomly picking instances from X without replacement, an Isolation Tree is built using this dataset and added to the set of trees built so far.

1.1.3 Anomaly Score and Path Length

After an Isolation Tree is built on a sub-sampled dataset, it is possible to measure the outlierness of an instance x and therefore it's anomaly score. As said before, in an Isolation Tree anomalies tend to be closer to the root of the tree in contrast to the inliers which end up at the deeper parts of the tree. This is depicted in figure 1.1 where the red path on the tree leads to the red leaf node, whereas the blue line goes to the deepest leaf node. The red leaf node is shorter than other paths on the tree. Therefore it has the highest anomaly score.

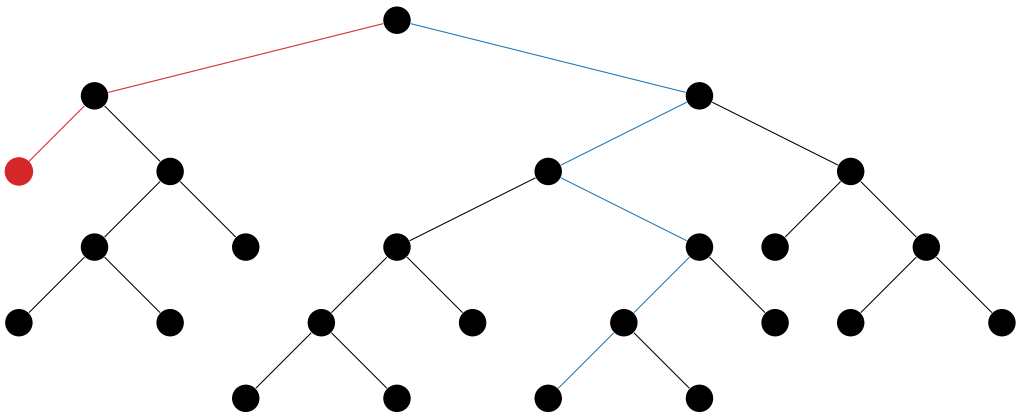


Figure 1.1: Representation of an Isolation Tree in an Isolation Forest

In order to evaluate anomaly score of an instance x we need to define *Path Length* $h(x)$ as the number of edges that x traverses from the root of the tree until it reaches an external node.

Defining the anomaly score of an instance based on it's path length $h(x)$ is difficult. Because even though that the maximum height of a tree increases in the order of n, it's average height grows in the order of $log\ n$. Therefore normalization of $h(x)$ is either not bounded or it can not be compared directly.

The structure of Isolation Trees are equivalent to *Binary Search Trees*. For this reason the estimation of average path length $h(x)$, is the same as the unsuccessful search in a Binary Search Tree. Given a dataset with n instances, [4] defines the average path length of unsuccessful search in a Binary Search Tree as follows:

$$c(n) = \begin{cases} 2H(n-1) - 2(n-1)/n & if\ n > 2, \\ 1 & if\ n = 2, \\ 0 & otherwise, \end{cases} \tag{1}$$

where $H(i)$ is the harmonic number and is estimated as:

$$H(i) \approx ln(i) + 0.5772156649. \tag{2}$$

Because $c(n)$ is the average path length of $h(x)$, we could use it to normalize $h(x)$. Therefore the anomaly score of an instance x would be defined as follows:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}, \tag{3}$$

where $E(h(x))$ is the average path length of x from an ensemble of trees. In this equation:

- If $E(h(x)) \to 0$, then $s \to 1$. If instances have anomaly score close to 1, then they are anomalies.

- If $E(h(x)) \to n - 1$, then $s \to 0$. If instances have anomaly score close to zero, then they are inliers.

In equation 3, $h(x)$ is the path length of an instance x on a single Isolation Tree. This value is calculated according to algorithm 3.

Algorithm 3: $PathLength(x, T, e)$

input : x - an instance, T - an iTree, e - current path length; to be initialized to zero when first called
output: path length of x

```
1 if T is an external node then
2 |  return e + c(T.size) {c(.) is defined in Equation 1}
3 end
4 a ← T.splitAtt
5 if X_a < T.splitValue then
6 |  return PathLength(x, T.Left, e + 1)
7 else
8 |  return PathLength(x, T.Right, e + 1)
9 end
```

Given an instance x and an Isolation Tree T, the *PathLength* function starts from the root node of the tree and counts the number of edges e that the instance x traverses until it reaches an external node. For external nodes with a $size > 1$ an adjustment value $c(T.size)$ is added to the number of edges traversed so far. This adjustment is the average path length of an unsuccessful search in an unbuilt subtree that has exeeded the height limit of the tree. By computing the path length $h(x)$ for all of the Isolation Trees in a forest, it's easy to calculate the average path length $E(h(x))$ and by plugging it into equation 3 the anomaly score of an instance x is calculated.

1.1.4 Complexity of the algorithm

The following table summarizes the time complexity and memory requirement of Isolation Forest for training stage as well as evaluation.

	Training	Evaluation
Time Complexity	$O(t \cdot \psi \cdot log\ \psi)$	$O(n \cdot t \cdot log\ \psi)$
Memory Requirement	$O(t \cdot \psi)$	-

Table 1.1: Overview of time complexity and memory requirement for Isolation Forest.

1.1.5 Discussion

Isolation Forest has unique characteristics that motivated us to improve it further as it is to be seen in the upcoming chapters.

- It has only two hyperparameters: number of trees in the ensemble t and the sub-sampling size ψ.
- Since each Isolation Tree uses a sub-sampled dataset, the algorithm does not need to see the entire data and therefore it is not required to load the entire dataset in memory.
- Not relying on any distance or density measure saves the algorithm a major computational cost and hence faster than distance/density based techniques.
- It is able to handle large and high dimensional datasets.
- Run time complexity of Isolation Forest is linear and it also has a low memory requirement.
- Like other Tree based machine learning algorithms there is no need to scale or normalize the input data.

1.2 Local outlier factor (LOF)

Local outlier factor (LOF) [5] is another classical unsupervised method to detect anomalies in a given dataset. Unlike Isolation Forest or other common methods in machine learning, LOF does not separate the train and test stage. That is, given a dataset, LOF assigns an anomaly score to each instance in a single run. In other words LOF assigns an outlier factor to each point. The outlier factor is local in the sense that only the surrounding neighborhood of each instance is considered for its calculation. This method shares some fundamental concepts with density-based clustering methods like [6].

1.2.1 Basic concepts and definitions

In order to calculate the local outlier factor of objects in a dataset, understanding the following concepts is necessary.

k-distance of an object p: For any positive integer k, the *k-distance* of an object p, denoted as *k-distance*(p), is simply the distance of that point and its k-th nearest neighbor. Formally, it is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that:

1. for at least k objects $o^{'} \in D \setminus \{p\}$ it holds that $d(p, o^{'}) \leq d(p, o)$,
2. for at most $k-1$ objects $o^{'} \in D \setminus \{p\}$ it holds that $d(p, o^{'}) < d(p, o)$.

k-distance neighborhood of an object p: The *k-distance neighborhood* of a point p is the set of all points in the dataset D, whose distances to the point p is less than or equal to *k-distance*(p). Formally,

$$N_{k\text{-}distance(p)}(p) = \{q \in D \setminus \{p\} \mid d(p, q) \leq k\text{-}distance(p)\}. \tag{4}$$

Such objects are called the k-nearest neighbors of the point p.

Reachability distance of an object p w.r.t. object o: For a positive integer k the *reachability distance* of object p with respect to object o is defined as follows:

$$reach\text{-}dist_k(p, o) = \max\{k\text{-}distance(o), d(p, o)\}. \tag{5}$$

That is, if p lies within the k-distance neighborhood of o, the reachability distance of p from o is the k-distance of o, otherwise it is simply the distance between them. For example in figure 1.2 for $k = 3$, $reach\text{-}dist_k(p_3, o)$ is the distance between o and p_3 whereas $reach\text{-}dist_k(p_1, o)$ is $k\text{-}distance(o)$.

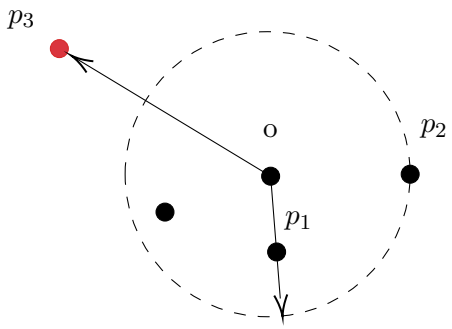


Figure 1.2: Illustration of reachability distance for $k = 3$

Local reachability density of an object p: It is the inverse of the average reachability distance based on the *MinPts*-nearest neighbors of p and is defined as follows:

$$lrd_{MinPts}(p) = \left(\frac{\sum_{o \in N_{MinPts}(p)} reach\text{-}dist_{MinPts}(p, o)}{\mid N_{MinPts}(p) \mid} \right)^{-1}. \tag{6}$$

(Local) outlier factor of an object p: It is the average of the ratio of the local reachability density of p and its $MinPts$-nearest neighbors and is defined as follows:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{\mid N_{MinPts}(p) \mid}. \tag{7}$$

In equation 7:

- when $LOF_{MinPts}(p) \approx 1$, then the point has a similar density as its neighbors;
- when $LOF_{MinPts}(p) < 1$, then the point is an inlier;
- when $LOF_{MinPts}(p) > 1$, then the point is an outlier.

1.2.2 Complexity of the algorithm

The LOF algorithm is a computationally intensive process. That is because it should compute the LOF of every object in the dataset and for that there are a large number of k-nearest neighbor queries required. That makes the time complexity of this algorithm $O(n^2)$. While it is possible to employ indexing methods to achieve a better run time complexity than a quadratic one, the effectiveness of such methods degrade with increasing dimensionality.

1.2.3 Discussion

An advantage of LOF is that due to its local approach it is able to detect anomalies in a dataset that would not be identified as outlier elsewhere in the same dataset. For example a single point near a dense cluster of points could be an outlier while another point in a sparse cluster of points that has larger distances to it's neighbors could be an inlier. This is depicted in figure 1.3.

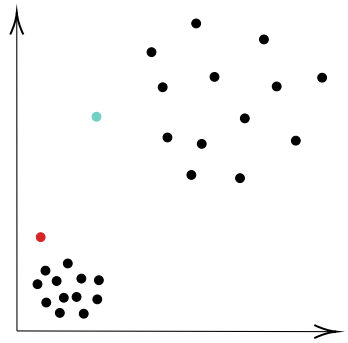


Figure 1.3: Illustration of the locality approach of LOF. The blue point can be considered as inlier although it is farther from the sparser cluster compared to the distance of the red point and its denser cluster.

A major drawback of LOF is that it's results are hard to interpret. The values less than 1 clearly inliers. But it is hard to say from which value (or threshold) a point is considered to be an outlier. For example a point with a value of 1.2 could be an outlier in some dataset while another point in another dataset could have a value of 2 and still be an inlier.

1.3 Autoencoder

The previous methods mentioned in this chapter to detect anomalies are types of classical machine learning algorithms [1]. On the other hand, the use of Autoencoders as a special type of neural networks is a modern technique to find anomalies. In recent years numerous types of neural networks has been developed that have different applications. This section begins with neural networks and how they are used to model non-linear patterns. After that autoencoders are introduced and how they are used to detect anomalies.

[1] Classical in the sense that they are non-neural network algorithms.

1.3.1 Neural Networks

Neural networks are a class of machine learning algorithms that mimic the behavior of the human nervous system. *Neurons* are special type of cells that can send or receive electrical signals. These signals travel through *synapses* to reach other neurons. If the electrical signals in a neuron exceed a certain threshold, this neuron would be activated and in turn sends an electrical signal to its neighbors. Although this is just a rough idea of how neurons in our nervous system work, it is sufficient for us to model *artificial* neural networks for the purpose of machine learning.

An artificial neural network is a directed graph where nodes are connected to each other with oriented edges. Like nervous system, a node in an artificial neural network receives input values from its incoming edges, produces an output value if the sum of its input values exceeds the threshold and passes this value to the next node via an edge. Each edge has a corresponding weight that is multiplied to the output of its preceding node. Figure 1.4 illustrates the simplest form of a neural network called a *perceptron*.

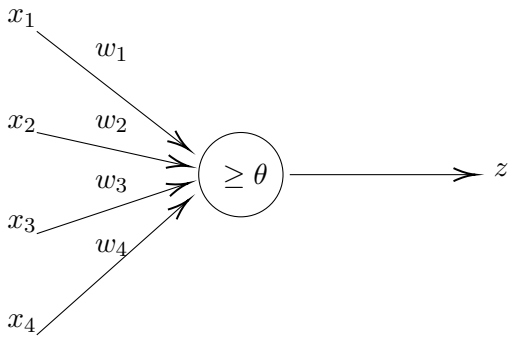


Figure 1.4: Illustration of a perceptron

Given a d-dimensional input $\overline{X} = (x_1, ..., x_d)$ and a vector of weights $\overline{W} = (w_1, ..., w_d)$, the output of a perceptron could be computed as follows:

$$z = \overline{W} \cdot \overline{X} = \sum_{i=1}^{d} w_i x_i. \tag{8}$$

This function is a linear activation function. In order to add non-linearity to the perceptron it is common to apply a non-linear function Φ to the output of the perceptron and also include a bias term θ:

$$z = \Phi(\overline{W} \cdot \overline{X} + \theta). \tag{9}$$

Examples of activation functions are:

$$\Phi_{(\text{Sigmoid function})}(z) = (1 + e^{-z})^{-1},$$
$$\Phi_{(\text{ReLU})}(z) = max\{0, z\},$$
$$\Phi_{(\text{Softplus})}(z) = log(1 + e^{z}).$$

Due to the nice conceptualization of a single neuron as a unit of computation, it is possible to put multiple layers of neurons together to form a multilayer neural network. This type of multilayer neural network is able to approximate any non-linear function. Therefore they are also called *universal function approximators* [7]. Figure 1.5 shows a typical architecture of a multilayer neural network with an input layer that corresponds to the shape of the input, two hidden layers and an output layer.

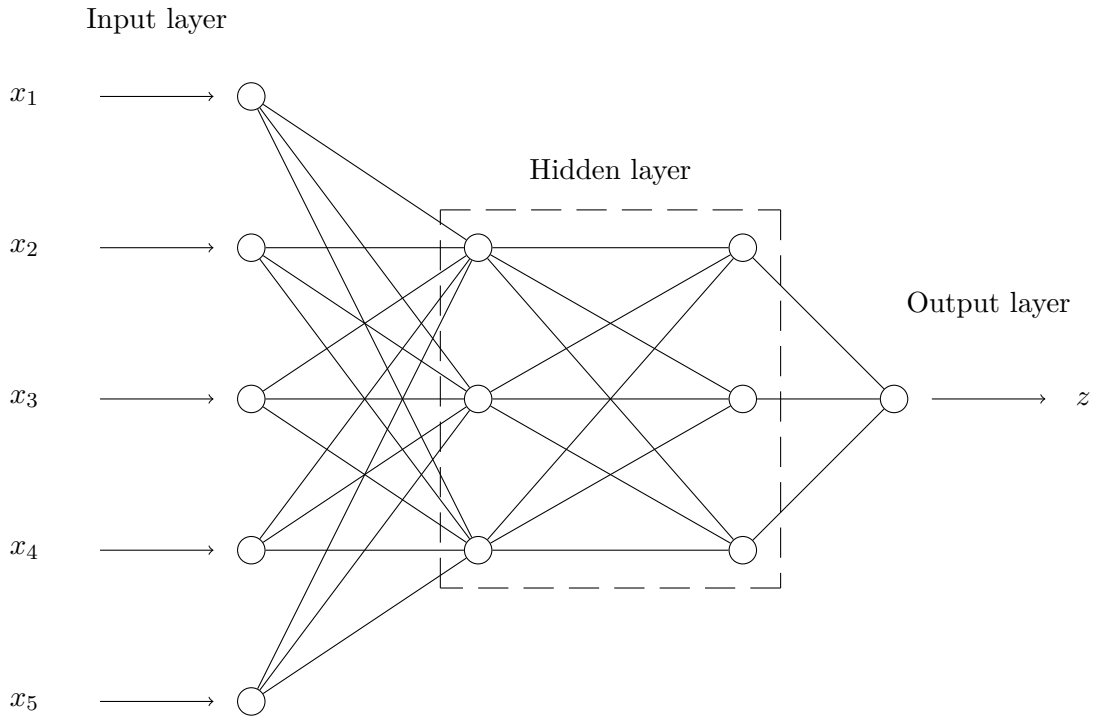


Figure 1.5: A multilayer neural network

By feeding the input layer with an input $\overline{X}$, the input values propagate through the network and produce output values in output layer.

Let $\hat{z}(\overline{X})$ be the output of a single perceptron and z be the correct value of the example. The error of the perceptron for that example is computed as follows:

$$E(\overline{W}) = (z - \hat{z}(\overline{X})) = (z - \Phi(\overline{W} \cdot \overline{X})). \tag{10}$$

As the weight vector $\overline{W}$ is the only variable in this equation, updating the weights of the

perceptron is the only way to account for this error. The new weight vector $\overline{W}$ is achieved with *gradient-decent*[1] as follows:

$$\overline{W} \Leftarrow \overline{W} - \eta \cdot \frac{\partial}{\partial \overline{W}} E(\overline{W})^2, \tag{11}$$

where η is the learning rate and the partial derivative of the squared error to $\overline{W}$ is derived as follows [2]:

$$\begin{aligned} \frac{\partial}{\partial \overline{W}} E(\overline{W})^2 &= 2 \cdot E(\overline{W}) \cdot \frac{\partial}{\partial \overline{W}} E(\overline{W}) \\ &= 2 \cdot E(\overline{W}) \cdot \frac{\partial}{\partial \overline{W}} (z - \Phi(\overline{W} \cdot \overline{X})) \\ &= -2 \cdot \overline{X} \cdot E(\overline{W}) \cdot \Phi'(\overline{W} \cdot \overline{X}). \end{aligned} \tag{12}$$

The process of gradient-decent for the entire network is repeated until some stopping criteria is reached, e.g. after a certain number of iterations.

1.3.2 Anomaly detection using Autoencoders

A special type of artificial neural networks is an autoencoder, which is trained to copy its input to its output. This is done by first encoding the input to a lower dimensional space representation (called a latent space), then decoding back the data from the latent space to reconstruct the input. This means that autoencoder learns to compress the data while minimizing the reconstruction error.

An autoencoder usually has a symmetric shape as figure 1.6 represents. One can split the network around the middle layer into two sides. The left side is an encoder and the right side is a decoder.

[1] Other than gradient decent there are other optimization algorithms as well, e.g. *Stochastic Gradient Descent*, *Adam*, etc.

[2] It is common to see that many optimize the squared error $\frac{1}{2} E(\overline{W})^2$ so that the constant cancels out the power of two after derivation. However, we have not done that according to equation 12 because the resulting constant could be considered as part of the learning rate η in equation 11 and hence be ignored.

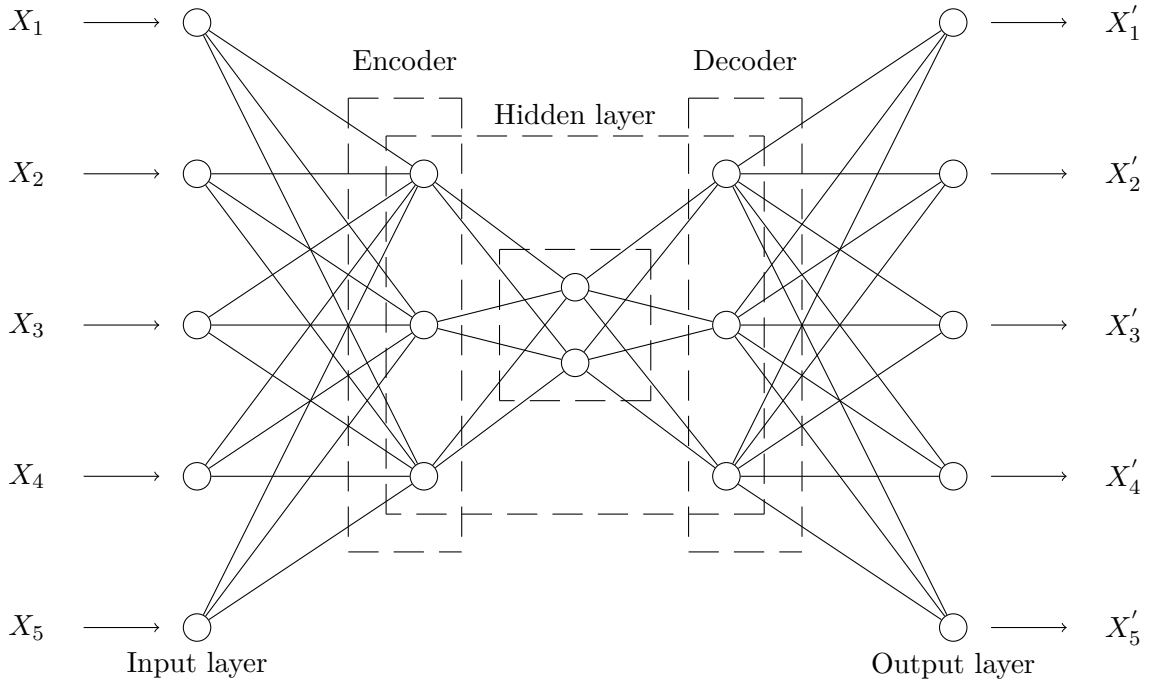


Figure 1.6: Illustration of an Autoencoder with three hidden layers

Among various applications of autoencoders like dimensionality reduction, feature extraction and image denoising, anomaly detection will be covered here. To perform anomaly detection the network should be trained on the normal instances of a dataset. The trained network has minimized its reconstruction error over the instances of the training dataset, i.e. normal data points. When the network is fed with an anomalous data point it usually has difficulties reconstructing the anomaly and therefore its reconstruction error will be high. For this reason the reconstruction error of the autoencoder is considered to be the measure of outlierness, i.e. anomaly score. Anomalies have a higher reconstruction error compared to the normal points, therefore they have higher anomaly scores.

1.3.3 Discussion

A neat characteristic of neural networks is their ability to cope with high dimensional data like images. This enables autoencoders to perform anomaly detection on such high dimensional data as well, whereas the majority of classical algorithms are not able to do so without a proper dimensionality reduction as a preprocessing step.

The downside of autoencoders, as a special type of neural networks, is that they are too slow to optimize and they need a large amount of data to be trained. Another problem with autoencoders is that they are sensitive to noise. Since anomalies are considered to be noises among the majority of the normal instances, training an autoencoder with a dataset that contains anomalies results in inaccuracies in the model. The reason is that autoencoder tries to reconstruct the anomalies during the training phase so that their reconstruction error would be minimized, which in turn assigns those points low anomaly scores.

2

Extending Isolation Forest to support categorical data

Chapter 1 introduced Isolation Forest as a method for anomaly detection. Like many other unsupervised anomaly detection methods, Isolation Forest is natively only compatible with numerical data. This leads the categorical features often be ignored in practice. Although it is possible to solve this issue by converting the categorical features into numerical ones using one-hot encoding, this has its drawbacks as well. Converting the categorical features using one-hot encoding often results in large vectors. Therefore during training and at each split, it is more likely that those one-hot encoded features be picked up in comparison to the numerical features. In other words, one-hot encoding puts more emphasis on converted categorical features than the original numerical ones.

Section 2.1 introduces an existing approach proposed by [8] to handle categorical data. After that in section 2.2 we generalize this approach and present our method. Later in chapter 4 we show that our generalization improves the algorithm.

2.1 Categorical data using a subset of length 1

As we saw earlier in chapter 1, an Isolation Tree picks a random split value at each split during the training stage (Algorithm 1). The assumption is that the attributes are all numerical. Authors of [8] propose a simple extension to this algorithm so that at each split the nature of the selected feature is determined. If it is a numerical feature the process is the same as before. Otherwise if the feature is categorical, an element is randomly picked from the set of possible values. The instances which are equal to the split value build the left subtree and the rest build the right subtree. Algorithm 4 presents this version.

Algorithm 4: *iTree(X, e, l)*

input : X - input data, e - current tree height, l - height limit
output: 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 $q \in Q$
  if *q is categorical* **then**
    randomly select a split value p in the domain of Q
    $X_l \leftarrow filter(X, q = p)$
    $X_r \leftarrow filter(X, q \neq p)$
  else
    randomly select a split point p from max and min values of attribute q in X
    $X_l \leftarrow filter(X, q < p)$
    $X_r \leftarrow filter(X, q \geq p)$
  end
  return $inNode\{Left \leftarrow iTree(X_l, e+1, l), Right \leftarrow iTree(X_r, e+1, l), SplittAtt \leftarrow q, SplitValue \leftarrow p\}$
end

Because this algorithm randomly picks only one element from the set of possible values, we refer to this approach for Isolation Forest on categorical data as *subset of length 1* [1].

2.2 Categorical data using a random subset

Our approach to handle categorical data is a generalization to the previous one with the difference that instead of picking a single element (or a subset of length 1) from the set of possible values of the categorical feature, a random subset p is chosen. The instances whose feature q has a value that is a member of set p go to the left subtree and the rest go to the right subtree. The only restriction for p is that the empty set and the original set itself are excluded [2]. This is depicted in algorithm 5.

[1] Note that a subset of length of 1 is equivalent to a single element.
[2] The reason is that theses two sets do not perform any split so that all the instances end up in one subtree.

Algorithm 5: *iTree(X, e, l)*

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

```
1  if e ≥ l or |X| ≤ 1 then
2      return exNode{Size ← |X|}
3  else
4      let Q be a list of attributes in X
5      randomly select an attribute q ∈ Q
6      if q is categorical then
7          S ← Set of possible values of attribute q
8          randomly select a subset p ⊂ S so that p ≠ ∅
9          X_l ← filter(X, q ∈ p)
10         X_r ← filter(X, q ∉ p)
11     else
12         randomly select a split point p from max and min values of attribute q in X
13         X_l ← filter(X, q < p)
14         X_r ← filter(X, q ≥ p)
15     end
16     return inNode{Left ← iTree(X_l, e + 1, l), Right ←
         iTree(X_r, e + 1, l), SplittAtt ← q, SplitValue ← p}
17 end
```

As before, we refer to this new approach for Isolation Forest on categorical data as *random subset*.

2.3 Discussion

Compared to Isolation Forest for categorical data using a *subset of length 1* our variation has the advantage that it makes more comparisons at each node. Being able to do more comparisons at each node reduces the overall number of nodes required in an Isolation Tree, which results in compactor and hence, smaller trees. Smaller trees are always preferable to larger ones since they take up less space and are faster to build. Isolation Forest for categorical data using a *subset of length 1* is equivalent to the scenario where one converts the categorical data with one-hot encoding into numerical ones and weights the features. This overcomes only the problem where categorical features are more likely to be picked up compared to the original ones. Therefore this method does nothing more than encapsulate the pre-processing step so that the user of the algorithm does not have to encode the categorical features. In chapter 4 we thoroughly examine these two variations using a synthetic dataset.

3

Extending Isolation Forest to support text data

In chapter 2 we explained our way of extending the Isolation Forest to support categorical data. In this chapter we use the same concept to extend the Isolation Forest even further to support textual data. To the best of our knowledge it is the first solution being proposed to detect anomalies in text data using Isolation Forest. First it is required to define some terms from the field of natural language processing (NLP) that are used throughout this chapter.

Stop words: Stop words are any words that occur so frequently in a text such that they have no significant semantic relation to the context in which they exist [9]. Therefore these words will be removed from the text before the processing of natural language. Examples are *to*, *and*, *the*, etc.

***n*_gram**: A successive collection of items in a text document is called an *n*_gram. An item could be a letter or a word. Possible word *n*_grams from the sentence *"The train was late."* are as follows:

- 1_gram: the, train, was, late
- 2_gram: the train, train was, was late
- 3_gram: the train was, train was late
- 4_gram: the train was late

Algorithm 6: *iTree(X, e, l, τ, **t)*

input : X - input data, e - current tree height, l - height limit, τ - Jaccard similarity threshold of two sets, *$*t$ - A dictionary of parameters to extract n_grams

output: 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 $q \in Q$
  if *q is textual* **then**
    $S \leftarrow$ extract n_grams from the entire corpus according to the parameters in *$*t$
    randomly select a subset $p \subset S$ so that $p \neq \emptyset$
    $X_l \leftarrow filter(X, J(q_{n\_grams}, p) \geq \tau)$
    $X_r \leftarrow filter(X, J(q_{n\_grams}, p) < \tau)$
  else if *q is categorical* **then**
    // same as before
  else if *q is numerical* **then**
    // same as before
  end
  return $inNode\{Left \leftarrow iTree(X_l, e+1, l), Right \leftarrow iTree(X_r, e+1, l), SplittAtt \leftarrow q, SplitValue \leftarrow p\}$
end

As algorithm 6 describes, when training an Isolation Tree at each split the type of the attribute q is determined. If it is textual a set S of possible n_grams from the entire q is extracted according to the parameters in *$*t$. These parameters are as follows:

- **splitter**: A regular expression determining the boundaries of a token. In other words everything that does not belong to the token is described by this regular expression.
- **n_gram_range**: A range of possible n_grams. For example the range (1, 3) allows 1-grams, 2-grams and 3-grams in the resulting set.
- **max_df**: Tokens with a higher frequency in the corpus than this given threshold are ignored.
- **min_df**: Tokens with a lower frequency in the corpus than this given threshold are ignored.
- **max_features**: If specified, considers only the first *max_features* top tokens according to their frequency in the entire corpus.
- **lowercase**: A boolean indicating whether to convert all characters to lowercase before tokenization.
- **stop_words**: A list of tokens that are not allowed in the final set of tokens.

In the next step a random subset of possible n_grams as the split value p is picked. The

empty set and S itself are excluded. The instances of the dataset are then divided into left and right subtrees according to their *jaccard similarity* of their set of $n_$grams to p:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{13}$$

The $n_$grams of each instance of the dataset should be extracted the same way that $n_$grams are extracted from the entire corpus. For example if a special character like "&" is considered to be a stop word, then it should be excluded from q_{n_grams} as well as S.

3.1 Discussion

We believe that there is no anomaly detection algorithm that can handle text directly without any pre-processing on the data and converting it to a numerical representation. Together with our extension of Isolation Forest to support categorical data one can use this algorithm on mixed data. In practice there are lots of data containing textual attributes. Since this algorithm does not capture the meaning of the underlying text, it's generally suitable for simple and semi-structured text data. One example is anomaly detection in HTTP traffic where the training data contains semi-structured texts like *url* or *user agent*.

With respect to its novelty one could still improve this solution. For example by replacing the jaccard similarity function in equation 13 with a metric that exploits the similarity of two sets of $n_$grams based on the semantics of them. The restriction of the jaccard similarity in this context is that it only measures the intersection of two sets of $n_$grams and does not capture the meaning of those two sets. Finding such a function makes the algorithm able to handle more complicated text data.

4

Experimental results

Chapter 1 introduced Isolation Forest, Autoencoder and LOF as three anomaly detection algorithms. In chapter 2 we saw two extensions for Isolation Forest to support categorical data, i.e. *subset of length 1* and *random subset*. In section 4.1 of this chapter the performance of these methods are compared with each other on 13 public datasets in terms of the area under the ROC curve. Later in section 4.2 we compare the two variations of Isolation Forest for categorical data, that is *subset of length 1* and *random subset* on a synthetic dataset.

4.1 Public datasets

The public datasets in this section are commonly used for anomaly detection and benchmarking of the algorithms and are available from the UCI machine learning repository [10]. Table 4.1 describes these datasets. Note that the majority of the datasets with the exception of "Chess", "Car" and "Mushroom" contain both categorical and numerical attributes.

Datasets	Samples	Attributes		Outliers	Description
		Num.	Cat.		
Cover Type	286.048	10	2	0.9%	Class 4 (anomalies) vs. class 2
German Credits	1.000	7	13	20%	Class bad (anomalies) vs. good
Chess	4.580	0	6	0.5%	Class zero (anomalies) vs. class fourteen
Car	1.275	0	6	5%	Class v-good (anomalies) vs. class unacc
Arrhythmia	420	205	73	13%	Classes 3, 4, 5, 7, 8, 9, 14, 15 (anomalies) vs. rest
Credit	653	6	9	45%	Class positive (anomalies) vs. rest
Adult	48.842	6	8	24%	Income > 50K (anomalies) vs. rest
Mammography	830	3	2	48%	Class malignant (anomalies) vs. benign
Lymphography	148	3	15	4%	Classes 1, 4 (anomalies) vs. rest
KDD_10%	494.021	33	7	80%	Class normal (inliers) vs. rest (anomalies)
KDD_SMTP	96.554	29	6	1%	Class normal (inliers) vs. rest (anomalies)
KDD_HTTP	623.091	28	5	0.6%	Class normal (inliers) vs. rest (anomalies)
Mushroom	5.644	0	22	38%	Class poisonous (anomalies) vs. edible

Table 4.1: Descriptions of public datasets

First, we want to evaluate the performance of anomaly detection methods on these datasets using only numerical attributes. Then we include the categorical attributes and perform a second experiment on these datasets. The performance of each method on each dataset is measured by the *area under the ROC curve* (ROC AUC). We compare the overall performance of anomaly detection methods over the entire datasets according to the average of their ROC AUCs and study the effect of including categorical attributes on the performance of the anomaly detection methods.

In order to measure the performance of each algorithm on a single dataset, we run it 10 times with different initial states [1] and report the average of ROC AUCs. It is not the case for LOF because this algorithm neither has any internal states nor any steps which requires randomness. We remove instances with missing values in any of their attributes from the datasets and also discard the attributes that have more than 10% missing values as suggested by the authors of [11]. Furthermore, the numerical attributes are all normalized in the range of $[0, 1]$. For Autoencoder and LOF we have converted categorical features

[1] An initial state defines the internal state of a model before the training is started. An example is the random assignment of initial weights and biases of neurons in a neural network. In addition, the random sub-sampling of training instances in Isolation Forest is also dependent on the initial state.

using one-hot encoding into their numerical representations since they don't support them. The algorithms are trained on normal instances and tested against the entire dataset. The choice of parameters for the algorithms are as follows:

- **Isolation Forest**: The number of estimators is set to 100 and the number of samples per tree is set to 256 as suggested by its authors [1].
- **Autoencoder**: Authors of [12] suggest the following values:
  - **Hidden layers**: Four hidden layers, where the layers have 64, 32, 32, and 64 neurons respectively from left to right. However the shape of the hidden layers depend on the number of attributes of the given dataset. For instance if a dataset has 30 attributes we design the structure of hidden layers as follows: 30, 15, 15, 30.
  - **Activation function for hidden layers**: ReLu,
  - **Activation function for the output layers**: Sigmoid.
- **LOF**: As suggested by [12] the number of neighbors is set to 20 and the distance function is *euclidean*.

Table 4.2 summarizes the results of running anomaly detection methods on public datasets, excluding the categorical attributes. The best result for each dataset is written in bold.

Dataset	Isolation Forest	Autoencoder	LOF
Cover Type	0.8641±0.049	0.9601±0.0074	**0.9931**
German Credit	0.5623±0.0059	0.5632±0.0028	**0.6079**
Chess	NA	NA	NA
Car	NA	NA	NA
Arrhythmia	**0.8306±0.018**	0.8227±0.000005	0.8176
Credit	**0.8790±0.0073**	0.7827±0.0017	0.7727
Adult	**0.6615±0.0069**	0.6199±0.014	0.6503
Mammography	0.7602±0.010	0.7039±0.018	**0.7905**
Lymphography	0.8665±0.013	**0.8680±0.0064**	0.8257
KDD_10%	0.9650±0.023	**0.9956±0.0000015**	0.8155
KDD_SMTP	**0.9988±0.00024**	0.9987±2.4e-06	0.9924
KDD_HTTP	0.9964±0.0025	**0.9999±8e-17**	0.9953
Mushroom	NA	NA	NA
Average AUC:	0.8384	0.8315	0.8261

Table 4.2: ROC AUC of different algorithms on public datasets using only numerical attributes

Note that the results for "Chess", "Car" and "Mushroom" datasets are not available (NA) since they don't contain any numerical attributes. The last row of the table provides the average of ROC AUCs for each algorithm over the entire datasets ignoring the ones with NA as a result. In terms of the average of ROC AUCs, Isolation Forest and Autoencoder perform almost the same, and they are both better than LOF. With regard to the number

of best results for each algorithm, Isolation Forest has the best performance on four datasets whereas Autoencoder and LOF have the best results on 3 three datasets each.

In the second round of experiments we included the categorical features and ran the algorithms on the public datasets. Table 4.3 shows the results. In figure 4.1 the two experiments on public datasets are compared with each other with regard to the average of their ROC AUCs over the entires datasets. Based on the average of ROC AUCs Isolation Forest with a *random subset* performs best overall. Also, in terms of the number of best results for each method, our variation of Isolation Forest is the best choice of algorithms for five datasets, whereas LOF, Autoencoder, and Isolation Forest with a *subset of length 1* are the best choices of algorithms for four, three and one datasets respectively. As figure 4.1 shows, including categorical features improves the performance of Autoencoder and the two variations of Isolation Forest. Table 4.4 shows the results of Autoencoder and LOF on public datasets side by side excluding and including categorical features. When categorical features are included, the performance of Autoencoder increases for five out of ten datasets containing mixed data, for three datasets it decreases, and for the rest two datasets, the performance doesn't change. Also, the average ROC AUC increases 2.7%. In the case of LOF we see a performance increase only in three datasets and the average ROC AUC decreases 11%. The effect of including the categorical attributes for both variations of Isolation Forest is shown in table 4.5. From seven out of ten datasets we see an improvement of the ROC AUC when the categorical attributes are involved for Isolation Forest with *random subset* and the average of ROC AUC increases 3.1%. While for five out of ten datasets we see an improvement for Isolation Forest with a *subset of length 1* and even though the average ROC AUC increases 1.8%, our method benefits more from categorical attributes on seven dataset.

Dataset	Isolation Forest		Autoencoder	LOF
	Random subset	Subset of length 1		
Cover Type	0.9785±0.014	0.9106±0.026	0.9754±1.7e-07	**0.9863**
German Credit	**0.6216±0.0098**	0.6142±0.010	0.6142±8.2-05	0.6076
Chess	0.9889±0.0089	0.8988±0.028	0.9944±1.1-16	**0.9948**
Car	0.9947±0.0038	0.9960±0.0036	**0.9999±7.6e-06**	0.5
Arrhythmia	**0.8318±0.011**	0.8244±0.011	0.7910±3.8e-05	0.7751
Credit	0.8658±0.0059	**0.8737±0.011**	0.8006±0.00017	0.7670
Adult	**0.6141±0.015**	0.5958±0.017	0.6107±6.2e-06	0.4967
Mammography	**0.8131±0.0095**	0.8038±0.014	0.8119±0.00045	0.5744
Lymphography	0.9980±0.0019	0.9821±0.0073	0.9976±0	**1.0**
KDD_10%	0.9830±0.010	0.9776±0.014	**0.9921±2.7e-08**	0.9624
KDD_SMTP	**0.9989±0.00027**	0.9981±0.00029	0.9987±7.9e-08	0.9936
KDD_HTTP	0.9939±0.0069	0.9905±0.011	**0.9999±7.8e-17**	0.9891
Mushroom	0.9957±0.0012	0.9735±0.0047	0.8658±5.1e-05	**0.9999**
Average AUC:	0.8983	0.8800	0.8809	0.8189

Table 4.3: ROC AUC of different algorithms on public datasets using numerical and categorical attributes

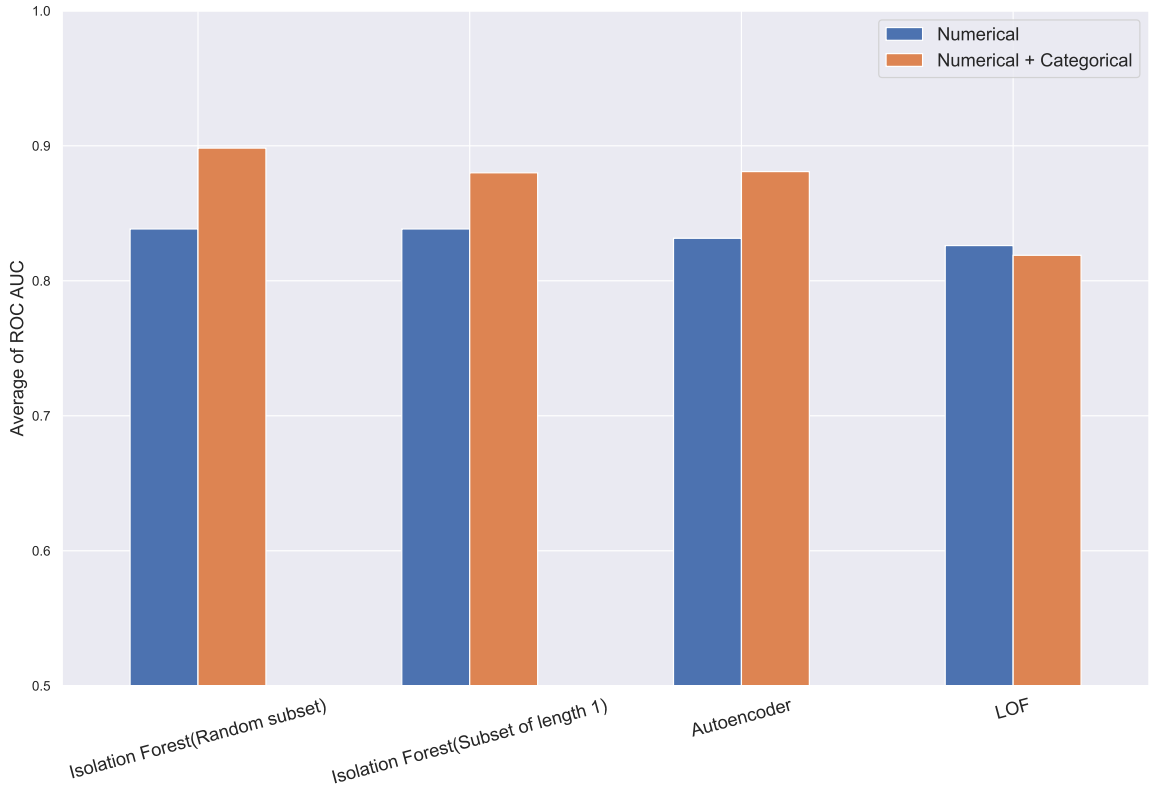


Figure 4.1: Comparisons of average ROC AUCs over the entire public datasets excluding and including the categorical features.

	Autoencoder		LOF	
Dataset	Numerical Data	Mixed Data	Numerical Data	Mixed Data
Cover Type	0.9601±0.0074	**0.9754±1.7e-07**	**0.9931**	0.9863
German Credit	0.5632±0.0028	**0.6142±8.2-05**	**0.6079**	0.6076
Arrhythmia	**0.8227±0.000005**	0.7910±3.8e-05	**0.8176**	0.7751
Credit	0.7827±0.0017	**0.8006±0.00017**	**0.7727**	0.7670
Adult	**0.6199±0.014**	0.6107±6.2e-06	**0.6503**	0.4967
Mammography	0.7039±0.018	**0.8119±0.00045**	**0.7905**	0.5744
Lymphography	0.8680±0.0064	**0.9976±0**	0.8257	**1.0**
KDD10	**0.9956±0.0000015**	0.9921±2.7e-08	0.8155	**0.9624**
KDD_SMTP	**0.9987±2.4e-06**	**0.9987±7.9e-08**	0.9924	**0.9936**
KDD_HTTP	**0.9999±8e-17**	**0.9999±7.8e-17**	**0.9953**	0.9891
Average AUC	0.8315	**0.8592**	**0.8261**	0.7152

Table 4.4: Comparison of performance of Autoencoder and LOF with and without categorical attributes

	Isolation Forest on numerical data	Isolation Forest on mixed data	
Dataset	-	Random subset	Subset of length 1
Cover Type	0.8641±0.049	**0.9785±0.014**	0.9106±0.026
German Credit	0.5623±0.0059	**0.6216±0.0098**	0.6142±0.010
Arrhythmia	0.8306±0.018	**0.8318±0.011**	0.8244±0.011
Credit	**0.8790±0.0073**	0.8658±0.0059	0.8737±0.011
Adult	**0.6615±0.0069**	0.6141±0.015	0.5958±0.017
Mammography	0.7602±0.010	**0.8131±0.0095**	0.8038±0.014
Lymphography	0.8665±0.013	**0.9980±0.0019**	0.9821±0.0073
KDD10	0.9650±0.023	**0.9830±0.010**	0.9776±0.014
KDD_SMTP	0.9988±0.00024	**0.9989±0.00027**	0.9981±0.00029
KDD_HTTP	**0.9964±0.0025**	0.9939±0.0069	0.9905±0.011
Average AUC:	0.8384	0.8698	0.8570

Table 4.5: Comparision of Isolation Forest on numerical data and mixed data

4.2 Synthetic dataset

Last section empirically showed that Isolation Forest surpasses an state of the art algorithm like Autoencoder both in terms of ROC AUC and the absolute number of best results on 13 real world datasets. In this section, we are going to focus only on the two variations of Isolation Forest, namely *random subset* and *subset of length 1*, and emphasize the strength of our algorithm compared to *subset of length 1* using a synthetic dataset which has only categorical attributes. Our artificial dataset consists of 9000 normal samples and 1000 anomalies and has 10 attributes. Table 4.6 illustrates an examples of our synthetic dataset.

Id	Feature 1	Feature 2	Feature 3	Feature 4	Class
1	A	S	R	O	0
2	E	K	U	A	0
3	F	B	V	C	0
4	R	I	N	B	0
5	Q	D	J	A	0
6	I	F	J	H	0
7	A	A	I	M	0
8	B	R	C	T	0
9	Y	X	Z	X	1
10	X	X	Y	Z	1
11	Z	Z	Y	Y	1
12	X	Z	Y	Y	1

Table 4.6: An example of the synthetic dataset

In this dataset anomalies are those points whose features have values that are not present among the normal data points. For example, the last four instances in the above table are anomalies because their features have X, Y and Z as values that are not present among the normal instances. There are 15 possible values for each feature of the inliers. For both variations of Isolation Forest, that is *random subset* and *subset of length 1*, we use an ensemble of 100 trees and 256 samples per tree. We use training datasets with different proportions of anomalies in them ranging from 0% to 10% in the whole training dataset. Finally we test the algorithm on the entire dataset. Figure 4.2 shows the performance of the two algorithms on the synthetic dataset when they are trained on datasets with different proportions of anomalies.

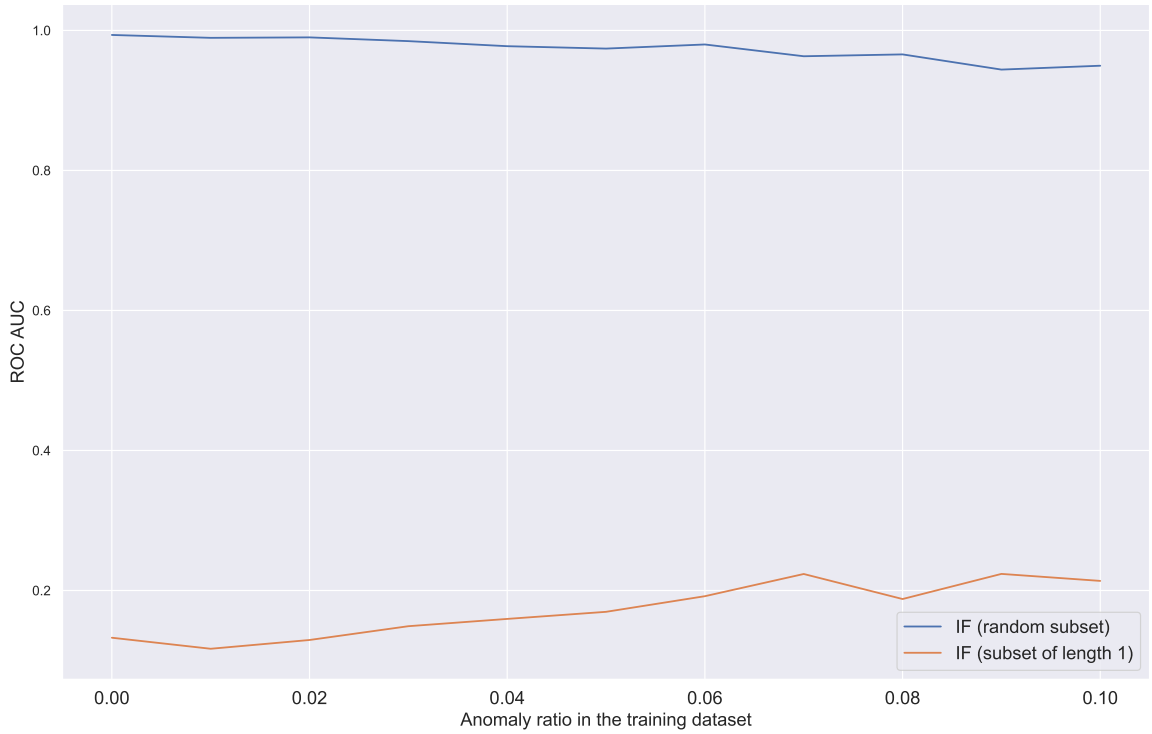
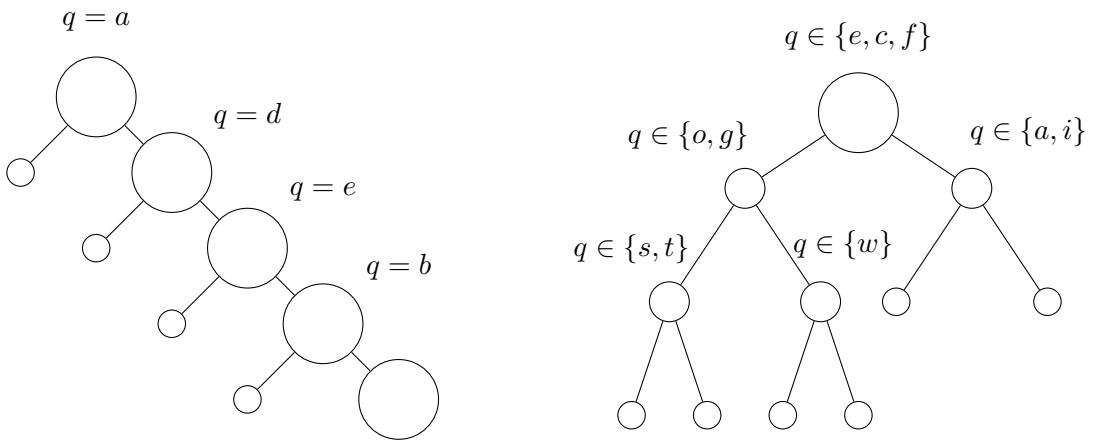


Figure 4.2: Comparison of two variations of Isolation Forest when trained on the synthetic dataset with different proportions of anomalies.

The reason that Isolation Forest with a *subset of length 1* performs so poorly on the synthetic dataset is explainable via the shape of the Isolation Trees. As we discussed in chapter 2, Isolation Forest with a *subset of length 1* splits the dataset by comparing the the values of feature q with a single value p. During the training stage, the majority of the samples do not fulfill the test criteria and therefore end up in the right child node. This situation occurs at each internal node resulting in an Isolation Tree that is unbalanced towards the right (figure 4.3(a)). The tree grows up until it reaches its depth limit leaving the majority of the dataset in a leaf node. When we use this tree for testing, the anomalies end up almost always in the right child nodes since their features have values that are not equal to any of the split values of the internal nodes. For this reason, anomalies traverse the unbalanced tree until the deepest leaf node, where the path length is the highest, and hence the anomaly score is the lowest. On the other hand, at each internal node, there are normal instances that fulfill the test criteria and go to a left subtree. Since the left subtrees did not have enough samples in them, the tree is not grown there, and therefore the path length of these samples is pretty short. Because of that, many of the normal instances will be misclassified as anomalies. On the contrary, Isolation Forest with *random subset* is not prone to this situation because at each internal node the split value is not restricted to a single element, but rather it is a set of elements. As a result more elements can satisfy the test condition and the tree will be more balanced (figure 4.3(b)).



(a) Unbalanced tree

(b) Balanced tree

Figure 4.3: A schematic comparison of a single Isolation Tree trained on our synthetic dataset using Isolation Forest with *subset of length 1* (left) and *random subset* (right).

4.3 Discussion

In section 4.1 we conducted two rounds of experiments on public datasets. In the first experiment, we used only the numerical features, and in the second one, we included the categorical features. The first experiment showed that the performance of Isolation Forest is as good as a deep learning method, namely Autoencoder, and even slightly better. The second set of experiments has two implications. The first one is that the Isolation Forest with a *random subset* outperforms all of the other algorithms. The second implication is that our variation of Isolation Forest benefits the most when categorical features are included. In other words, Isolation Forest with a *random subset* is the best choice of algorithms on mixed datasets.

In section 4.2 we focused on the two variations of Isolation Forest using a synthetic dataset containing only categorical features. With this dataset we highlighted the weakness of Isolation Forest with a *subset of length 1* where it is outperformed by Isolation Forest with a *random subset*.

These results confirm the improvement of our Isolation Forest over the one proposed by [8].

5

Web Application Firewall

A *Web Application Firewall* (WAF) is responsible to detect and block harmful requests in HTTP-traffic. A WAF works as a reverse proxy and before the incoming requests reach their destination they are scanned by the WAF. Typically the WAF has a set of pre-defined rules which define the harmful requests. A request will be blocked if it matches one of the rules defined in the WAF. These rules are created by humans, and therefore there is no guarantee that they cover all of the possible malicious requests and attacks. In order to find new unusual activities that were not modeled before one can use machine learning. This chapter shows how we used Isolation Forest in practice to tackle this problem at the company *Continentale Krankenversicherung a.G.*

5.1 WAF Dataset & experimental setups

The training dataset that we use for this purpose contains the requests from the beginning of November 2020 until the end of April 2021 and is 420 gigabytes in size. There are 18 attributes as the table 5.1 shows. The last feature in the table, namely *Hour*, is derived from the time stamp and was not originally contained in the dataset. For training we do not use the Time and the response features, which are in italics. Response features contain the decision of WAF and therefore should not be used. Note that other than Hour other features are either categorical or textual. Using methods that can only handle numerical data adds a lot of unnecessary pre-processing efforts. For this reason, we use our variation of Isolation Forest since it can handle categorical and textual features inherently. Another reason to use Isolation Forest is because of its sub-sampling property. Clearly it is not feasible to load 420 GB of data into memory at once. Therefore we sampled this dataset by randomly picking instances without replacement and created a new dataset. The resulting dataset is only about 400 megabytes in size and has nearly 100000 instances and therefore can fit into memory easily.

Feature name	Type	Example value
Time	DateTime	2020-11-01 14:34:12
context.reverseProxyName	Categorical	RP_Strict
context.tunnelName	Categorical	Prod-Continentale
port	Categorical	443
request.ipSrc	String	45.43.184.11
request.method	Categorical	Post
request.path	Text	/html/js/barebone.jsp
request.protocol	Categorical	HTTP/1.1
request.query	Text	?browserId=other&languageId=de_D
response.backendResponseTime	Numeric	46322.0
response.totalResponseTime	Numeric	137795.0
response.size	Numeric	17098.0
response.statusCode	Categorical	404
security.cipher	Categorical	ECDHE-RSA-AES128-GCM-SHA256
security.protocol	Categorical	TLSv1.2
referer.path	Text	https://www.continentale.de/
referer.query	Text	/persoenliche-beratung?vepId=93514
user-agent	Text	Java/14.0.2
Hour*	Numeric	14

Table 5.1: An overview of the attributes of the Web Application Firewall

For training we use an Isolation Forest with an ensemble of size 100 and a sub-sampling size of 256. The features we used for training are Hour (Numeric), request.method (categorical) and five string features, namely request.path, request.query, referer.path, referer.query, and user-agent. The parameters for each string feature are shown in the table 5.2.

	String features parameters					
String feature	splitter(RegEx)	n_gram_range	max_df	min_df	max_features	lowercase
request.path	/	(1, 2)	0.65	0	200	False
request.query	?=&	(1, 2)	0.65	0	200	False
referer.path	/	(1, 2)	0.65	0	200	False
referer.query	?=&	(1, 2)	0.65	0	200	False
user-agent	&=?/(;:.)\]\[\d\s	(1, 2)	0.65	0	200	True

Table 5.2: Parameters for each string feature

Note that the splitter is specific to each string feature. For example, in the path part of a URL a forward slash (/) separates two tokens from each other and acts as a word boundary, whereas in the query part of the URL, separators are "&", "=" and "?", and hence splitter in our case.

We build models on different *tunnels* (context.tunnelName) using only its normal instances and test the model on the entire instances of that tunnel. Tunnel is another name for the application.

5.2 Experiments

Table 5.3 shows the performance results of each model trained and tested on different tunnels. For each tunnel the ROC AUC together with the performance metrics of the model with the optimal threshold are reported. Since Isolation Forest assigns an anomaly score to each instance that ranges from 0 to 1, we have to choose a threshold value so that the higher scores become anomalies and lower ones become inliers. To do so, we consider all of the distinct assigned anomaly scores returned by the algorithm. We iterate this list and each time set the threshold to one of the values and calculate the true positive rate and the false positive rate. True positive rate is defined as:

$$TPR = \frac{TP}{TP + FN}, \tag{14}$$

and the false positive rate is:

$$FPR = \frac{FP}{FP + TN}. \tag{15}$$

We then define the optimal threshold as the one that maximizes the difference between the true positive rate and the false positive rate. Formally,

$$optimal\_threshold = thresholds(argmax(TPR - FPR)). \tag{16}$$

The last row of the table 5.3 shows the average of different performance metrics over the entire models. Since anomalies, i.e. the positive class are rare, good metrics to evaluate the performance of our models are precision, recall, and their harmonic mean F1 Score [13]. Together with accuracy theses four metrics are defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}, \tag{17}$$

$$Precision = \frac{TP}{TP + FP}, \tag{18}$$

$$Recall = \frac{TP}{TP + FN}, \tag{19}$$

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{20}$$

An average recall of 0.88 means that the models can detect the harmful requests to a great extent. Such requests are the ones that the WAF can also identify. However, we would like to know whether there are other types of requests that the WAF has overlooked. For this reason, we expect our models to produce false positives and therefore have low precision. These models are considered to be pessimistic and are not suitable to replace a WAF because

		Performance of the best model						
TunnelName	ROC AUC	Accuracy	Recall	Precision	F1 Score	Predicted outliers	Actual outliers	Total instances
Prod-BiPro-Prod	0.6259	0.4028	1.0	0.0001	0.0003	11282	2	18890
Prod-Maklerportal	0.8848	0.7286	0.9345	0.0272	0.0530	3665	107	13166
Prod-Continentale	0.8960	0.7698	0.8879	0.0383	0.0735	2683	116	11267
Prod-KundenportalEuGo	0.7290	0.9217	0.4705	0.0142	0.0275	563	17	7204
Prod-Mannheimer	0.9264	0.9073	0.8228	0.3285	0.4695	834	333	6682
Prod-OnlinerechnerContactM_176_32_40_35	0.9526	0.8153	1.0	0.0047	0.0095	837	4	4512
Prod-OnlinerechnerContactM_176_32_40_35	0.7467	0.9672	0.75	0.0208	0.0405	144	4	4339
Prod-KundenportalEuropa	0.6690	0.4880	1.0	0.0014	0.0029	2064	3	4026
Prod-Belmot	0.8306	0.7102	0.9714	0.0321	0.0621	1059	35	3541
Prod-Artima	0.8292	0.8069	0.9375	0.0476	0.0907	629	32	3113
Prod-ImSound	0.7012	0.6102	0.8846	0.0192	0.0376	1196	26	3017
Prod-Lumit	0.8044	0.6718	1.0	0.0264	0.0515	945	25	2804
Prod-Sinfonima	0.4105	0.1925	1.0	0.1239	0.0244	1775	22	2171
Prod-www2-Conti_176_32_40_4	0.5845	0.6745	0.625	0.0082	0.0163	605	8	1853
Prod-www2-Conti_176_32_40_5	0.7534	0.7406	0.7777	0.0148	0.0291	472	9	1801
Prod-KFZRechnerEuGo	0.7603	0.7605	1.0	0.0025	0.0050	393	1	1637
Prod-EuropaGo_176_32_40_73	0.8632	0.7374	1.0	0.0262	0.0511	381	10	1413
Prod-EuropaGo_176_32_40_72	0.8302	0.7123	0.85	0.0411	0.0785	413	20	1387
Prod-VermittlerMH	0.4293	0.5260	0.6	0.0054	0.0108	68	5	1152
Prod-Email-Continentale	0.9943	0.9906	1.0	0.4117	0.5833	17	7	1073
Prod-ESignatur	0.9503	0.8033	1.0	0.0336	0.0651	208	7	1022
Prod-Shop-Continentale	0.7855	0.7850	1.0	0.0090	0.0180	110	1	507
Average:	0.7707	0.7147	0.8869	0.0562	0.0818			

Table 5.3: Results of WAF

of their low precision which could block lots of harmless requests. In order to discover new types of anomalies that WAF has missed, we analyze the anomalies predicted by the Isolation Forest in an offline manner. The procedure of manually finding suspicious requests is quite exhaustive because of the large number of false positives and therefore needs automation which is out of the scope of this work.

We may now point out to the reader, what causes the difference between the results of WAF and Isolation Forest:

- **Missing features**: Two important features, namely *post payload data* and *cookies* are not available in the training dataset that we used, which could result in false negatives in the model's predictions. As a result many relative attacks like cookie poisoning [14] can not be detected with a model trained on this dataset.
- **Domain knowledge**: Certain users and IP addresses are already known to the WAF and are not considered as harmful regardless of their activity. An example is the *monitoring system* of the company, which simulates the behavior of a normal user. The monitoring system usually fails to simulate a rational user and its actions leads to lots of unusual but not harmful requests, which produce many false positives in the model's predictions.
- **Interpretation**: It is important to make a clear distinction between malicious and unusual requests. Although unusual or rare requests are anomalies, they are not necessarily harmful and do not need to be blocked. For example, an old browser sets the user-agent to a value that is not common, and therefore its requests would be identified as anomalies by the model.

Nevertheless, we have analyzed the models of the tunnels "Prod-Email-Continentale" and "Prod-Mannheimer" with 0.41 and 0.32 precision, respectively. We found some suspicious requests that were not detected by the WAF. As table 5.4 shows, for the first six requests the user-agents "OpenVAS" and "cyberscan.io" are suspicious and therefore should be blocked since these two programs are used to perform penetration testing. Normal though the last request seems, its request.query is suspicious. In WordPress [1] an attacker can find the username by appending a query like "/?query=1" to their request, and the username is then revealed in the response to that query [2]. Furthermore, we also checked the IP addresses of these requests, and they are coming from China, which adds another reason to our suspicion about them.

[1] https://wordpress.org/
[2] https://www.wp-tweaks.com/hackers-can-find-your-wordpress-username

Id	context.tunnelName	request.path	request.query	user-agent
1	Prod-Email-Continentale	/webmailer/docs/docs/struts2-core-apidocs/index-all.html		Mozilla/5.0 [en] (X11, U; OpenVAS-VT 9.0.3)
2	Prod-Email-Continentale	/webreports/		Mozilla/5.0 [en] (X11, U; OpenVAS-VT 9.0.3)
3	Prod-Email-Continentale	/telescope/		Mozilla/5.0 [en] (X11, U; OpenVAS-VT 9.0.3)
4	Prod-Mannheimer	/index.html		Mozilla/5.0 [en] (X11, U; OpenVAS-VT 9.0.3)
5	Prod-Mannheimer	/index.html		Mozilla/5.0 [en] (X11, U; OpenVAS-VT 9.0.3)
6	Prod-Mannheimer	/customviews/image/login_bg/		cyberscan.io
7	Prod-Mannheimer	/	?author=2	Apache-HttpClient/4.5.2 (Java/1.8.0_151)

Table 5.4: Suspicious requests found in the tunnels "Prod-Mannheimer" and "Prod-EMail-Continentale"

5.3 Challenges and future works

The recall of the Isolation Forest on the WAF dataset indicates that it can detect harmful requests to a great extent. We even have managed to find suspicious requests during our manual analysis that WAF has missed. As we said at the beginning of this work, the motivation behind extending the Isolation Forest to support non-numerical features was the existence of numerous categorical and textual features in the WAF dataset. However, the work of anomaly detection in HTTP traffic does not stop here and further work is required. In our case:

- We need a better data acquisition process so that the features like ”cookies” and ”post payload data” are included in the training dataset while maintaining information security and privacy.
- More domain knowledge should be accessible to the model. For example, the requests of the ”monitoring system” should be removed from the dataset.
- Employing feature engineering techniques like the Wrapper method [15] helps to find the combination of features that result in the best performance.

Finally, an interesting question left unanswered is, whether the models trained on a specific tunnel could be used for prediction on the data of another tunnel. In other words, could we train a model once and use it generally for every scenario which involves detecting suspicious requests in HTTP traffic?

6

Conclusion & Outlook

Anomaly detection is the process of finding rare and unusual data points that deviate from the rest of the data so that they often raise suspicion. Despite their rareness, anomalies are present in nearly every dataset and as such finding them is an important topic in machine learning. Detecting unusual activities in computer network traffic, finding rare diseases in medical records of patients and determining fraud in trading activities or financial transactions are examples of anomaly detection. The focus of this work is primarily on Isolation Forest, how we have extended it to support non-numerical features, and how we used this algorithm in practice at the *Continentale Krankenversicherung a.G.*

Chapter 1 was a survey on three common anomaly detection algorithms, namely Isolation Forest, Local Outlier Factor (LOF) and Autoencoder:

- An Isolation Forest is a collection of binary trees referred to as Isolation Trees. For each split in an Isolation Tree both the feature and its split value are chosen randomly. In each Isolation Tree, the data points that get isolated with fewer splits are more anomalous than the others.
- LOF is a proximity-based algorithm where the outlierness of a data point is measured by its local deviation from its neighbors.
- An Autoencoder is a special type of neural network with a symmetric shape that consists of two components: An encoder and a decoder. The Autoencoder is trained to copy the input to the output layer by first encoding the input data to a lower dimensional space and decoding it back to reconstruct the input. For the purpose of anomaly detection, the loss function measures how much a point deviates from the normal data.

Like many other anomaly detection methods, these three algorithms support intrinsically only numerical attributes.

In chapter 2 we looked at solutions to generalize Isolation Forest so that it can support categorical features. The first solution was proposed by [8]. Their idea is, if the split

encounters a categorical feature, an element is randomly chosen from the possible set of values of the selected feature as the split value. The instances whose corresponding feature has the exact same value go to the left branch of the tree and the rest go to the right branch. Since the split value is a single element we refer to this approach as Isolation Forest with a *subset of length 1*. The second solution is our idea that generalizes the other approach. At each split, if the selected feature is categorical a random subset is chosen as the split value instead of a single element. Because in this approach the split value is a set the split is done by testing whether the corresponding feature of the instances has a value that belongs to the split value. Similarly, we called our solution Isolation Forest with a *random subset*.

In chapter 3 we used our idea of handling categorical features to extend Isolation Forest even further and support textual data. At each split if the selected feature is textual, a set of n_grams from the entire corpus is extracted. Then as the split value, a random subset of n_grams is chosen. An instance goes to the left subtree if the set of extracted n_grams of its textual feature is similar enough to the split value. We use the jaccard distance to measure the similarity between the two sets.

We compared the performance of our variation of Isolation Forest on 13 public datasets with Autoencoder and LOF in Chapter 4. In the first round of experiments, we used only the numerical features of those public datasets. Our observation was that in terms of ROC AUC, Isolation Forest and Autoencoder perform almost the same and they are both better than LOF. In the second round of experiments, we included the categorical features and saw that Isolation Forest with a *random subset* outperforms other algorithms. Importantly, Isolation Forest with a *random subset* benefits the most from the categorical features. Later in the chapter, we used our synthetic dataset to highlight the difference and the strength of our method over the Isolation Forest with a *subset of length 1*.

In chapter 5 we used our version of Isolation Forest that supports categorical and textual features to detect malicious requests in HTTP traffic at the *Continentale Krankenversicherung a.G.* A Web Application Firewall (WAF) monitors the incoming HTTP requests and blocks them when they match one of its predefined rules. This is a perfect application for our algorithm as the majority of the WAF's features are either categorical or textual. Furthermore, since Isolation Forest uses sub-sampling we did not need to use the entire training data, therefore we sampled the WAF's dataset. We aimed to analyze the predictions of the Isolation Forest in an offline manner and define new rules for the WAF when we find new types of malicious requests. The results of our experiments on the WAF dataset were models with high recalls, meaning that the models could find the harmful requests that the WAF detects to a great extent. The models had low precisions, which was a result of the false positives as we expected. Although we detected new types of suspicious requests by our manual analysis, we still need to automate this process. Further future works in this topic are (a) a dataset with two important features that are currently missing, namely "cookies" and "post payload data", (b) making the domain knowledge of the WAF accessible to the models, and (c) using feature engineering techniques to find the most useful features for training. It is also necessary to distinguish harmful requests from uncommon but harmless ones.

The interesting properties of Isolation Forest make it open for improvements such as our extensions to support categorical and textual features. Other improvements are for example:

- **Interpretable Anomaly Detection with DIFFI**: proposed by [16] to make predictions of Isolation Forest interpretable.
- **IForestASD: Isolation Forest Algorithm for Stream Data Method**: proposed by [17] to detect anomalies in data streams where the concept drift occurs [18].
- **Extended Isolation Forest**: proposed by [19] to split data according to a randomly chosen hyperplane that is not necessarily parallel to a single axis.
- **Active anomaly detection**: [20] proposes a technique to employ Isolation Forest for active anomaly detection in which the human feedback is used to improve the prediction of the algorithm.

We finish this work by proposing a new idea to tweak this algorithm further. It would be interesting for us to know, whether it is possible to achieve better results with numerical data if we handle them analog to categorical data. Our idea is that instead of picking a random value between *min* and *max* at a numerical split, some non-overlapping intervals be selected. The split is done by querying the membership of the numbers to one of these intervals. The immediate benefit of this improvement would be Isolation Trees that carry out more decisions in each internal node, which makes them more compact, as we saw in section 4.2. This raises further questions such as:

- How many intervals are required?
- How can we generate random intervals from the continuous uniform distribution?
- Does it bring any advantage in terms of ROC AUC over an approach where we treat the numerical features as categorical ones and apply the Isolation Forest for categorical features with *random subset* on them?

To answer these questions, we need a dedicated work to perform extensive experiments on public datasets containing numerical features, as we did in this work.

List of Figures

List of Tables

Bibliography

1. Liu, F. T., Ting, K. M. & Zhou, Z.-H. Isolation Forest, 413–422 (2008).
2. Breiman, L. Random Forests, 5–32 (2001).
3. Knuth, D. E. *The art of Computer Programming: Volume 3: Sorting and Searching.* ISBN: 978-0201896855 (Addison Wesley, 1998).
4. Preiss, B. R. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java* ISBN: 978-0-471-34613-5 (Wiley, 1999).
5. Breunig, M. M., Kriegel, H.-P., Ng, R. T. & Sander, J. LOF: Identifying Density-Based Local Outliers. *ACM SIGMOD,* 93–104 (2000).
6. Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, 226–231 (1996).
7. Aggarwal, C. C. *Outlier Analysis second edition* 98, 110. ISBN: 978-3-319-47577-6 (Springer, 2017).
8. Garchery, M. & Granitzer, M. On the influence of categorical features in ranking anomalies using mixed data, 77–86 (2018).
9. Alajmi, A., Saad, E. M. & Darwish, R. R. Toward an ARABIC Stop-Words List Generation. *International Journal of Computer Applications* (2012).
10. *UCI Machine Learning Repository* https://archive.ics.uci.edu/ml/index.php. Accessed: 2022-03.
11. Campos, G. O. *et al.* On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery volume 30,* 891, 927 (2016).
12. Zhao, Y., Nasrullah, Z. & Li, Z. PyOD: A Python Toolbox for Scalable Outlier Detection. *Journal of Machine Learning Research* **20,** 1–7. http://jmlr.org/papers/v20/19-011.html (2019).
13. SUN, Y., WONG, A. K. C. & KAMEL, M. S. CLASSIFICATION OF IMBALANCED DATA: A REVIEW. *International Journal of Pattern Recognition and Artificial Intelligence,* 687–719 (2009).
14. Awad, M., Ali, M., Takruri, M. & Ismail, S. Security vulnerabilities related to web-based data. *TELKOMNIKA (Telecommunication Computing Electronics and Control),* 852–856 (2019).
15. Baek, S.-H., Kim, B.-M., Kang, B.-Y. & Kim, H.-G. Prognosis Prediction for Class III Malocclusion Treatment by Feature Wrapping Method, 684 (2009).
16. Carletti, M., Terzi, M. & Susto, G. A. Interpretable Anomaly Detection with DIFFI: Depth-based Isolation Forest Feature Importance (2020).

17. Ding, Z. & Fei, M. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window. *IFAC Proceedings Volumes,* 12–17 (2013).
18. Zhou, F. *et al.* Anomaly detection over concept drifting data streams. *Journal of Computational Information Systems,* 1697–1703 (2009).
19. Hariri, S., Kind, M. C. & Brunner, R. J. Extended Isolation Forest (2018).
20. Das, S., Islam, M. R., Jayakodi, N. K. & Rao, D. J. Active Anomaly Detection via Ensembles: Insights, Algorithms, and Interpretability (2019).

Eidesstattliche Versicherung

(Affidavit)

Barghidarian, Sina

Name, Vorname
(surname, first name)

209053

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Extending Isolation Forest to support non-numerical data

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, den 29.03.2022

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, den 29.03.2022

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**