# Isolation Forests

Tanner Brown

April 29, 2024

## 1 Introduction

Isolation Forest is an unsupervised learning algorithm that uses many binary-trees to detect anomalies. This is done in two phases: the training stage and the evaluation stage. In the training stage, we create many binary-trees by randomly and recursively partitioning our dataset. In the evaluation stage, we pass an instance into our forest and use the average depth of our instance to give it an anomaly score.

## 2 The intuition behind isolation forests

Outliers differ from normal data points in two important ways. Firstly, they are numerically different from other data points. Secondly, there are few of them. Because of this, outliers require few "partitions" to be separated from the dataset. Figure 1 shows this concept off. Point A is clearly an outlier, and thus it only takes two partitions to separate it from the rest of the dataset. Comparatively, other points in the dataset, despite being partitioned, are still boxed with many other points.

## 3 Training Stage

The training stage consists of two algorithms: the algorithm for creating the isolation tree and the algorithm that creates the forest.

### 3.1 Creating the isolation tree

Isolation trees are composed of two different types of nodes: internal and external. Internal nodes have a left and right child node as well as a split attribute and split value. The split attribute is an attribute chosen randomly among the datasets attributes. The split value is a randomly generated number between the minimum and maximum value of the split attribute.

External nodes have no children. Rather, they have a size attribute determined by the size of the dataset at that node.

The algorithm (in pseudocode) looks like below:

X - dataset contained by an array or other data structure.

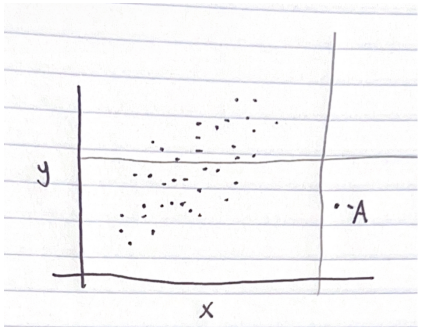


Figure 1: Example of partitioning on outliers.

h - the current height of the tree
l - the height limit of the tree
function *iTree*(X, h, l):
if (h $\geq$ l or |X| $\leq$ 1):
return exNode(size: |X|)
else:
splitAttr = random attribute on X
splitVal = random number between min and max of splitAttr
XLeft = filter(X, X $\leq$ splitVal)
XRight = filter(X, X > splitVal)
return inNode(left: iTree(XLeft, h+1,l), right: iTree(XRight, h+1, l),
splitAttr: splitAttr, splitVal: splitVal)

### 3.2 Creating the isolation forest

The isolation forest algorithm involves determining the height limit and determining the number of trees. Three variables are passed into the algorithm. The first is the dataset. The second is t, which determines the number of trees. The third is $\psi$, which determines the sub-sample size and height limit of the tree. We sub-sample the dataset to prevent swamping and masking. Masking is when outliers cluster together and become difficult to partition, and swamping is when outliers get too close to normal instances. Fei Tony Liu, who initially developed the algorithm, empirically found that t=100 and $\psi$=$2^8$ (or 256) work well, and through my own testing I can confirm that. The algorithm is as follows:
function *iForest*(X, t, $\psi$):
l = ceiling($\log_2\psi$)
for(i to t):
Xsample = sample(X, $\psi$)
Forest.add( iTree(Xsample, 0, l)
return Forest

## 4 Evaluation stage

Once we have our forest we can evaluate a new instance using it. Evaluation is done using a score ranged 0 to 1. Scores close to one are considered outliers, and scores close to 0 are considered normal instances. If all scores in the dataset are close to 0.5, then it is likely that the dataset contains no outliers. Equation 1 is what we use to find our score s. h(x) is the path length takes in a tree, and E(h(x)) is the average path length for all t trees. Equation 2 is the average path length of an unsuccessful search in a binary tree. Since our isolation trees are structured like binary trees, we use c(n) to normalize h(x). Equation 3 is the harmonic number necessary for calculating c(n), where $\gamma$ is the Euler-Mascheroni constant.
(1) s(x,n) = $2^{-E(h(x))/c(n)}$
(2) c(n) = 2H(n-1)-(2(n-1)/n)
(3) H(n) = ln(n) + $\gamma$
h(x) is found using our final algorithm:
x - an instance being evaluated
tree - an isolation tree being used to evaluate x
h - the current tree height
function *pathlength*(x, tree, h):
if(tree is an inNode):
a = tree.splitAttr
if(x[attr] < tree.splitVal):
return pathlength(x, tree.left, h+1)
else:
return pathlength(x, tree.right, h+1)
else:
if(T.size > 1):

```
        return h
    else:
        return h + c(T.size)
```

The pathlength function is applied on every tree for an instance, and we average all h(x) found by doing this to find E(h(x)).

# 5 Implementation

My implementation was done using Python. Aside from using the pandas, math, and random modules it was written from scratch. I tested it using a credit card fraud dataset from Kaggle. Only 492 of the 284,807 instances in the dataset are considered outliers. An unfortunate consequence of isolation forests is that it needs an input to give an anomaly score to, which is difficult when I don't have any new data to give it. To remedy this I find the anomaly score of a random row, of a randomly generated instance, of an outlier that is in the dataset, the min and max anomaly score in the dataset, and an average anomaly score. I find that the anomaly score ranges from approximately 0.35 to 0.80 with an average of approximately 0.43. The average being close to 0.5 lets us conclude that the dataset has few anomalies, which is corroborated by the aforementioned number of anomalies.

Isolation forest has a time complexity of $O(t\psi log\psi)$ in the training state, and an $O(ntlog\psi)$ time complexity in the evaluation stage; achieving a linear time complexity. Compared to other anomaly detection algorithms, isolation forests work fundamentally differently since it does not do computationally expensive distance calculations.

# 6 Conclusion

Isolation forests are an interesting take on isolating anomalies. Using the idea that outliers are few and different, isolation forests diverge from other anomaly detection algorithms by using partitioning rather than costly distance calculations. Because of this difference, isolation forests have a low memory requirement with a linear time complexity.

# References

"Isolation Forest." Wikipedia, Wikimedia Foundation, 11 Jan. 2024, en.wikipedia.org/wiki/Isolation_forest.
Liu, Fei Tony, and Kai Ming Tang. Isolation Forest, www.lamda.nju.edu.cn/publication/icdm08b.pdf. Accessed 27 Apr. 2024.

ULB, Machine Learning Group -. "Credit Card Fraud Detection." Kaggle, 23 Mar. 2018, www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data.