Anomaly detection using Isolation Forest – A Complete Guide

A akshara_416
Last Updated : 04 Apr, 2025

Anomaly detection is crucial in data mining and machine learning, finding applications in fraud detection, network security, and more. The Isolation Forest algorithm, introduced by Fei Tony Liu and Zhi-Hua Zhou in 2008, stands out among anomaly detection methods. It uses decision trees to efficiently isolate anomalies by randomly selecting features and splitting data based on threshold values. This approach is effective in quickly identifying outliers, making it well-suited for large datasets where anomalies are rare and distinct.

As in this article, we delve into the workings of the Isolation Forest algorithm, its implementation in **Python**, and its role as a powerful tool in anomaly detection. We also explore the metrics used to evaluate its performance and discuss its applications across various domains.

In this article, you will get a clear understanding of the Isolation Forest algorithm in Python. We will look at how to use Isolation Forest for finding outliers in data. You will learn about the Isolation Forest Python library and see an easy Isolation Forest example. By the end, you will know how to use isolation forest outlier detection in your

Learning Outcomes

- Understand the concept of Average Path Length (APL) in the context of **decision trees** and its relevance in anomaly detection.
- Explain the structure and functionality of **Binary Search Trees** (BSTs) and their application in organizing data for efficient search operations.
- Apply data analysis techniques to analyze and interpret data stored in a Pandas DataFrame.
- Discuss the significance of the Eighth IEEE International Conference on Data Mining (ICDM) and its contributions to the field of data science.
- Explore the contributions of Fei Tony Liu and Kai Ming to the development of the Isolation Forest (iForest) model for anomaly detection.
- Describe the principles and workings of the Isolation Forest model and its

- Evaluate the effectiveness of the Isolation Forest model using appropriate metrics and interpret the results for anomaly detection.

This article was published as a part of the Data Science Blogathon.

Table of contents

What is Isolation Forest?

Isolation Forest is a method used to find unusual data points, known as anomalies or outliers, in a dataset. It is particularly good at spotting these anomalies in large amounts of data.

Since its introduction, Isolation Forest has gained popularity as a fast and reliable

Isolation Forests(IF), similar to **Random Forests**, are build based on **decision trees**. And since there are no pre-defined labels here, it is an unsupervised model.

> “ Isolation Forests were built based on the fact that anomalies are the data points that are “few and different”.

In an Isolation Forest, randomly sub-sampled data is processed in a tree structure based on randomly selected features. The samples that travel deeper into the tree are less likely to be anomalies as they required more cuts to isolate them. Similarly, the samples which end up in shorter branches indicate anomalies as it was easier for the tree to separate them from other observations.

Let’s take a deeper look at how this actually works.

How do Isolation Forests work?

As mentioned earlier, Isolation Forests outlier detection are nothing but an ensemble of binary decision trees. And each tree in an Isolation Forest is called an Isolation Tree(iTree). The algorithm starts with the training of the data, by generating Isolation Trees.

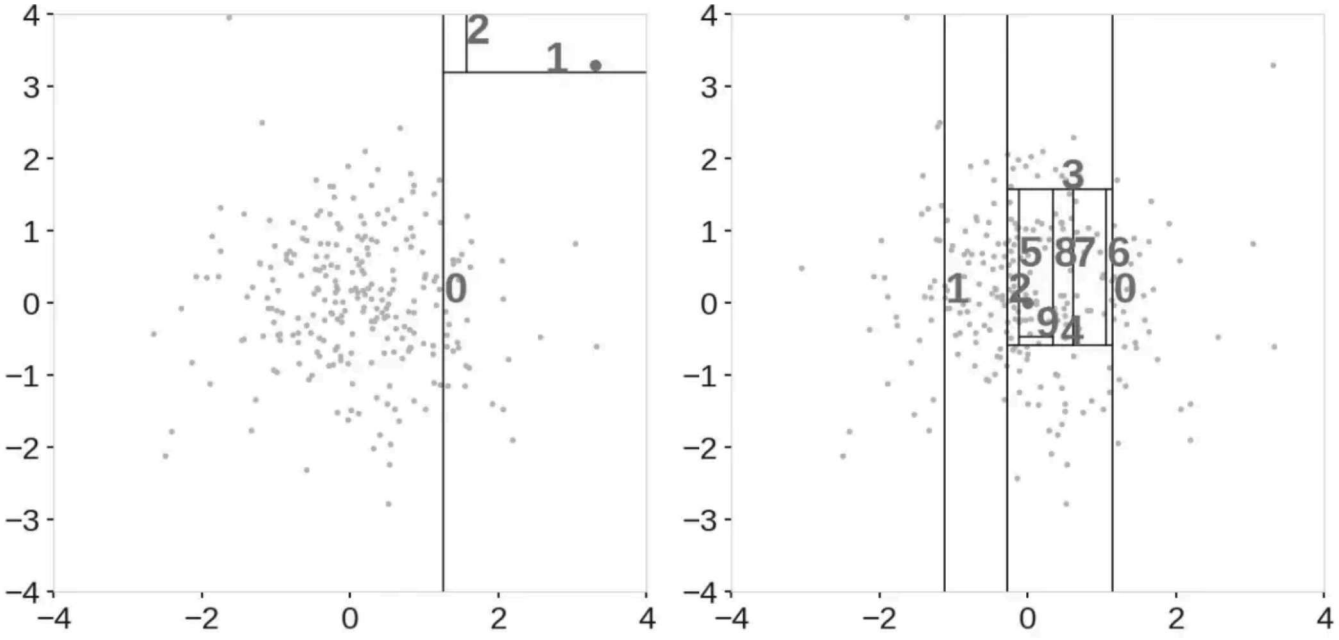
Step by Step Guide on How Isolation Forest Work

Let us look at the complete algorithm step by step:

- Branching of the tree starts by selecting a random feature (from the set of all N features) first. And then branching is done on a random threshold (any value in the range of minimum and maximum values of the selected feature).
- If the value of a data point is less than the selected threshold, it goes to the left branch else to the right. And thus a node is split into left and right branches.
- This process from step 2 is continued recursively till each data point is completely isolated or till max depth(if defined) is reached.
- The above steps are repeated to construct random binary trees.

After creating an ensemble of iTrees (Isolation Forest), the model training is complete. During scoring, the system traverses each data point through all the previously trained trees. The model assigns an ‘anomaly score’ to each data point based on the depth of the tree needed to reach that point. This score aggregates the depths obtained from each of the iTrees. An anomaly score of -1 assigns anomalies and 1 to normal points based on the contamination parameter (percentage of anomalies present in the data).

ReadMore about the Improve Dataset Selection with ChatGPT



(a) Anomaly point (b) Nominal point

Source: **IEEE**. We can see that it was easier to isolate an anomaly compared to a normal observation.

Implementation in Python

Let us look at how to implement Isolation Forest in Python.

Step1: Read the Input Data

Copy Code

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.ensemble import IsolationForest
data = pd.read_csv('marks.csv')
data.head(10)
```

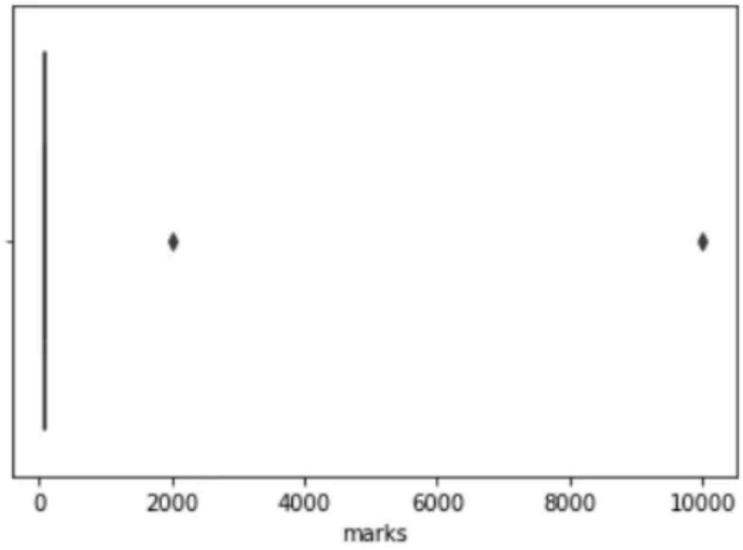
	student id	marks
0	1	95
1	2	98
2	3	92
3	4	10000
4	5	91
5	6	89
6	7	90
7	8	2000
8	9	100
9	10	100

Step2: Visualize the Data

Copy Code

```
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
data = pd.read_csv('marks.csv')
print(data.head(10))

sns.boxplot(data.marks)
plt.show()
```



From the box plot, we can infer that there are anomalies on the right.

Step3: Define and Fit the Model

```
random_state = np.random.RandomState(42)

model=IsolationForest(n_estimators=100,max_samples='auto',contamination=float(

model.fit(data[['marks']])

print(model.get_params())
```

Copy Code

Output:

```
{'bootstrap': False, 'contamination': 0.2, 'max_features': 1.0, 'max_samples': 'auto', 'n_
```

You can take a look at Isolation Forest documentation in sklearn to understand the model parameters.

```
data['anomaly_score'] = model.predict(data[['marks']])

data[data['anomaly_score']==-1].head()
```

Output:

	student id	marks	anomaly_score	scores
3	4	10000	-1	-0.330730
7	8	2000	-1	-0.210603

Here, we can observe that both anomalies are assigned an anomaly score of -1.

Step4: Model Evaluation

Copy Code

```
accuracy = 100*list(data['anomaly_score']).count(-1)/(anomaly_count)
print("Accuracy of the model:", accuracy)
```

Output:

```
Accuracy of the model: 100.0
```

What happens if we change the contamination parameter? Give it a try!!

Read More about the SVM One-Class Classifier For Anomaly Detection

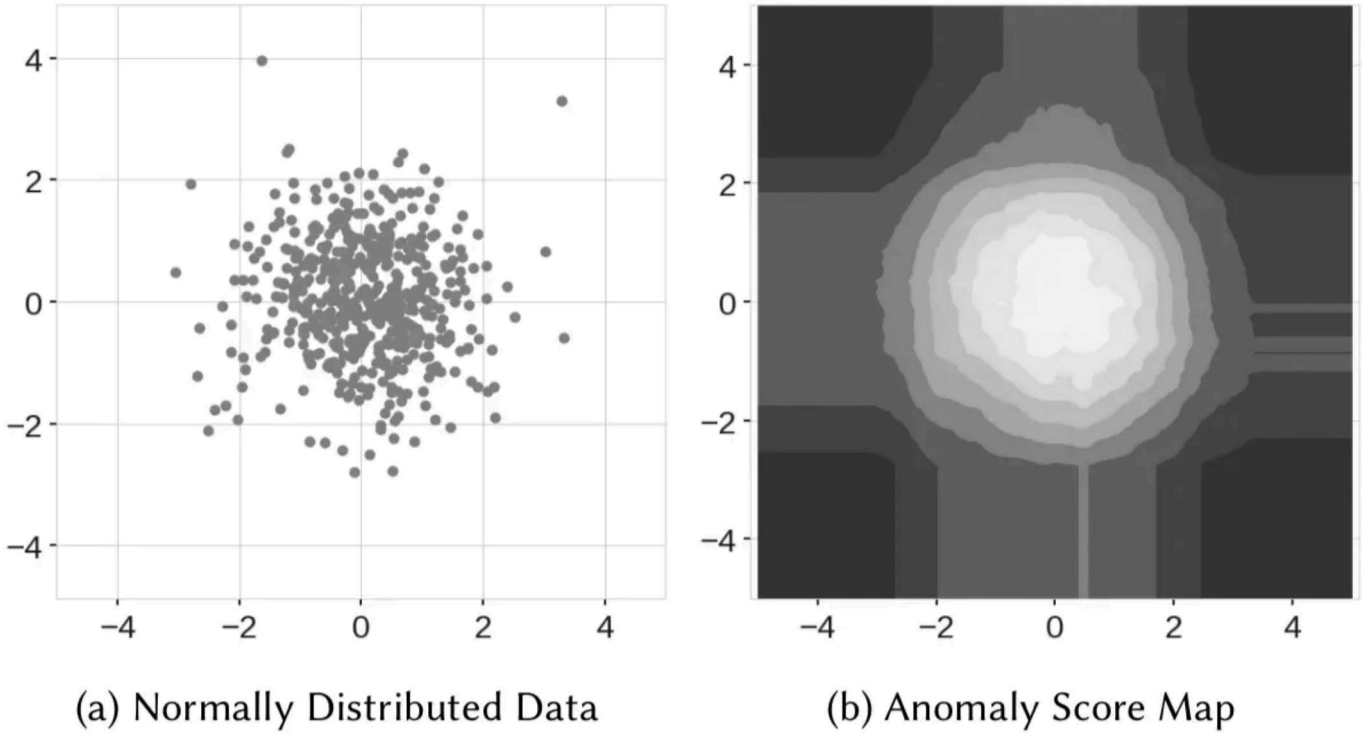
Limitations of Isolation Forest

Isolation Forests are computationally efficient and researchers have proven them to be

- The final anomaly score depends on the contamination parameter, provided while training the model. This implies that we should have an idea of what percentage of the data is anomalous beforehand to get a better prediction.
- Also, the model suffers from a bias due to the way the branching takes place.

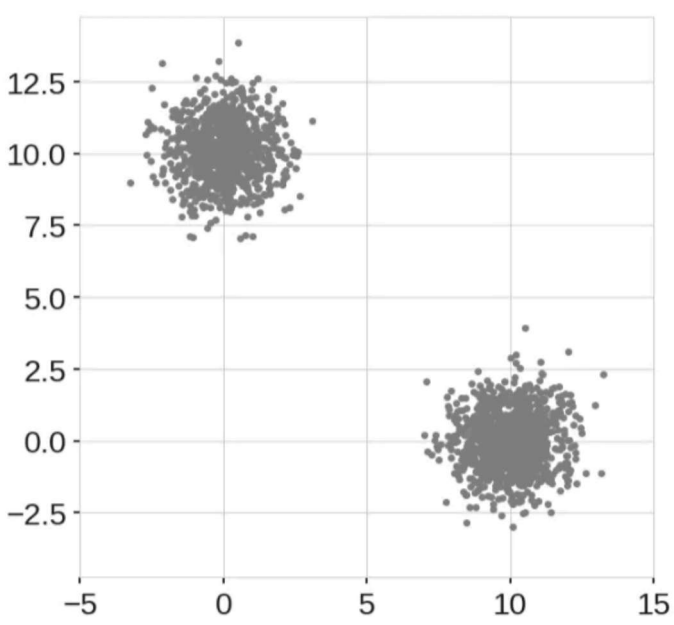
Anomaly Score Map

Well, to understand the second point, we can take a look at the below anomaly score map.

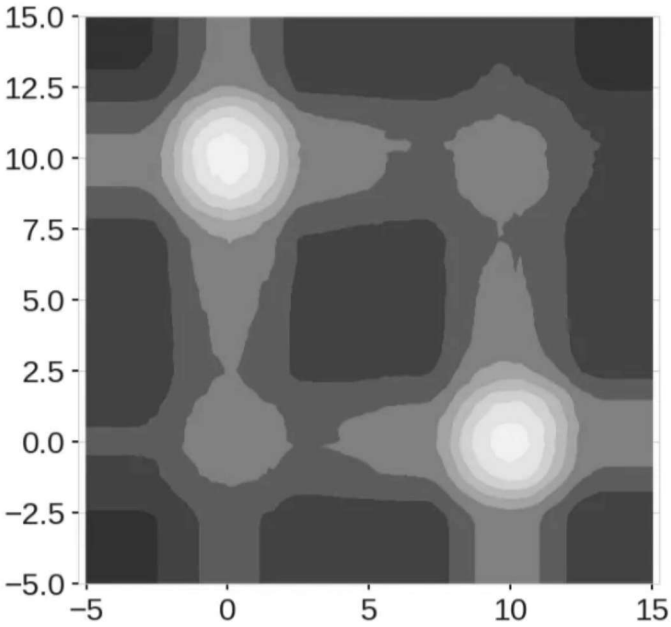


Source : IEEE

Here, in the score map on the right, we can observe that the points in the center obtained the lowest anomaly score, as expected. However, we can see four



(a) Two normally distributed clusters

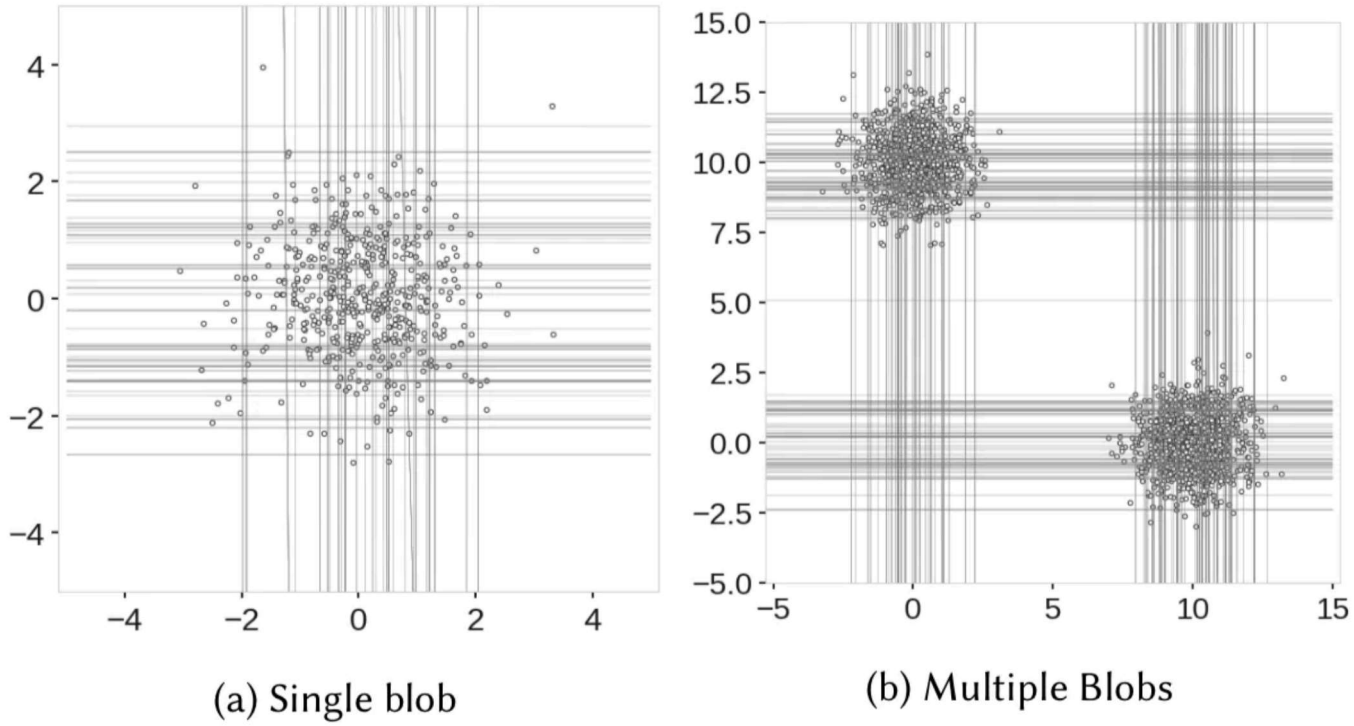


(b) Anomaly Score Map

Similarly, in the above figure, we can see that the model resulted in two additional blobs(on the top right and bottom left) which never even existed in the data.

Whenever a node splits in an iTree based on a threshold value, it divides the data into left and right branches, resulting in horizontal and vertical branch cuts. And these branch cuts result in this model bias.

Result

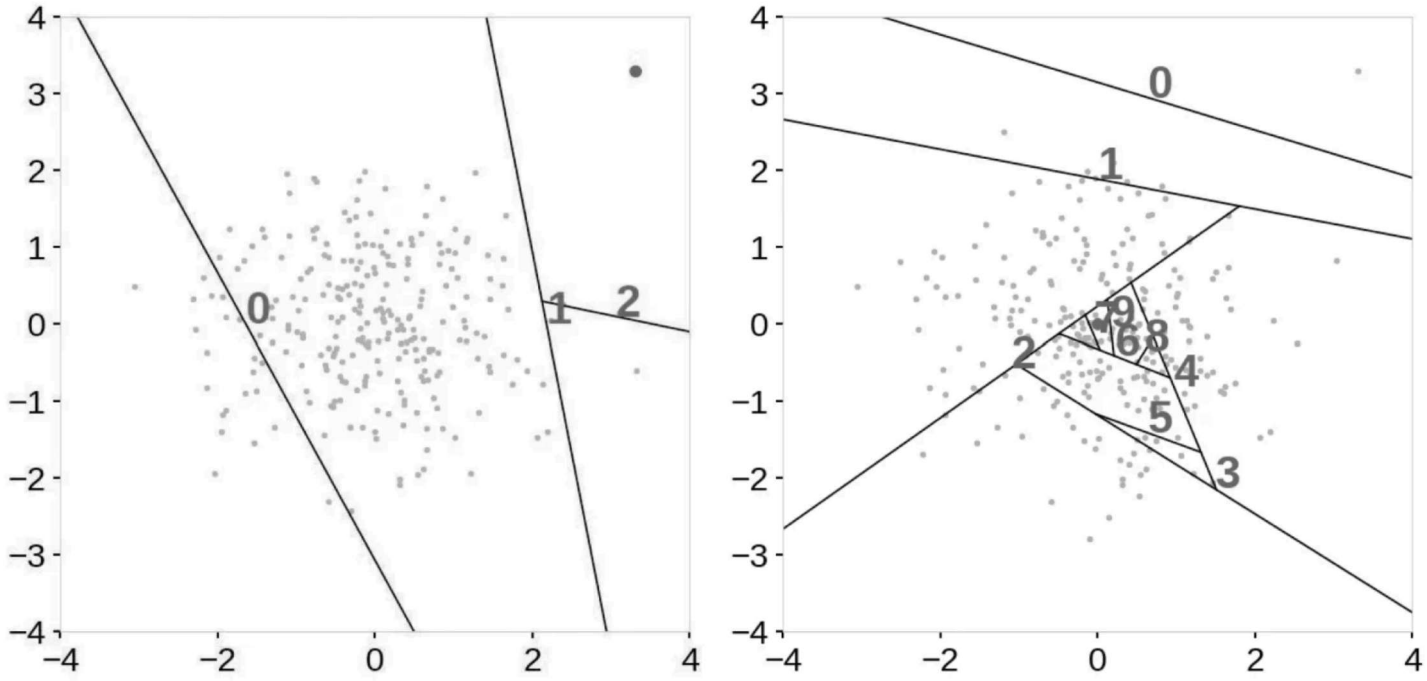


(a) Single blob

(b) Multiple Blobs

The above figure shows branch cuts after combining outputs of all the trees of an Isolation Forest. Here, we can observe how the rectangular regions with lower anomaly scores formed in the left figure. And also the right figure shows the formation of two additional blobs due to more branch cuts.

To overcome this limit, an extension to Isolation Forests called ‘Extended Isolation Forests’ was introduced by **Sahand Hariri**. In EIF, horizontal and vertical cuts were replaced with cuts with random slopes.



(a) Anomaly (b) Nominal

Despite introducing EIF (Extended Isolation Forest), the use of Isolation Forests for anomaly detection remains widespread across various fields.

Conclusion

The Local Outlier Factor (LOF) algorithm is a powerful tool for detecting anomalies in data by evaluating the density of points relative to their neighbors. Unlike traditional anomaly detection methods, LOF does not require assuming a specific data distribution, making it suitable for a wide range of applications. When implementing LOF, setting the n_estimators parameter appropriately ensures robust performance, balancing computational efficiency with detection accuracy. It is important to note that LOF performs well with both normal and anomalous instances, as it identifies points