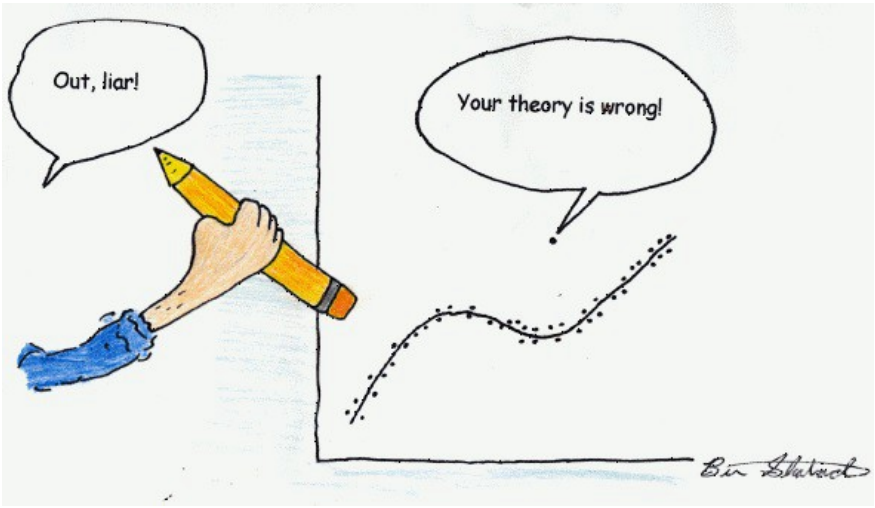

Outlier Detection with Isolation Forest

Eryk Lewinson Follow
Jul 2, 2018 · 6 min read ★

Update: Part 2 available here.

During a recent project I was working on a clustering problem with data collected from users of a mobile app. The goal was to classify the users in terms of their behaviour, potentially with the use of K-means clustering. However, after inspecting the data it turned out that some users represented abnormal behaviour—they were outliers.

A lot of machine learning algorithms suffer in terms of their performance when outliers are not taken care of. In order to avoid this kind of problems you could, for example, drop them from your sample, cap the values at some reasonable point (based on domain knowledge) or transform the data. However, in this article I would like to focus on identifying them and leave the possible solutions for another time.

As in my case I took a lot of features into consideration, I ideally wanted to have an algorithm that would identify the outliers in a multidimensional space. That is when I came across Isolation Forest, a method which in principle is similar to the well-known and popular Random Forest. In this article I will focus on the Isolation Forest, without describing in detail the ideas behind decision trees and ensembles, as there is already a plethora of good sources available.

Some theory first

The main idea, which is different from other popular outlier detection methods, is that Isolation Forest explicitly identifies anomalies instead of profiling normal data points. Isolation Forest, like any tree ensemble method, is built on the basis of decision trees. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature.

In principle, outliers are less frequent than regular observations and are different from them in terms of values (they lie further away from the regular observations in the feature space). That is why by using such random partitioning they should be identified closer to the root of the tree (shorter average path length, i.e., the number of edges an observation must pass in the tree going from the root to the terminal node), with fewer splits necessary.

The idea of identifying a normal vs. abnormal observation can be observed in Figure 1 from [1]. A normal point (on the left) requires more partitions to be identified than an abnormal point (right).

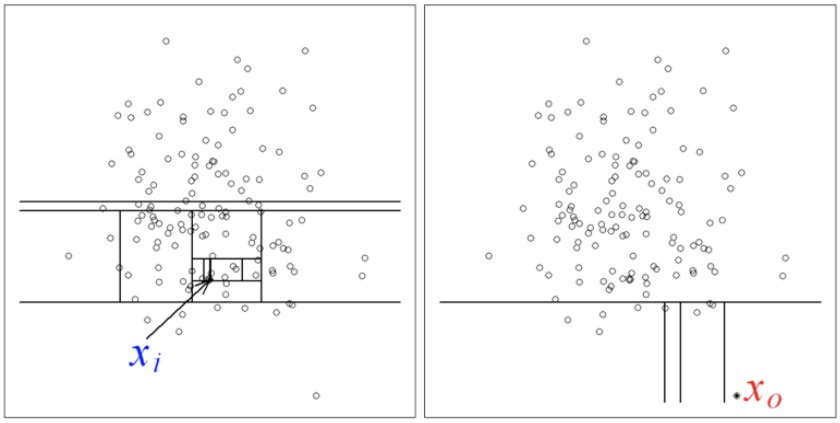

Figure 1 Identifying normal vs. abnormal observations

As with other outlier detection methods, an anomaly score is required for decision making. In case of Isolation Forest it is defined as:

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

where $h(x)$ is the path length of observation x, $c(n)$ is the average path length of unsuccessful search in a Binary Search Tree and n is the number of external nodes. More on the anomaly score and its components can be read in [1].

Each observation is given an anomaly score and the following decision can be made on its basis:

- Score close to 1 indicates anomalies
- Score much smaller than 0.5 indicates normal observations
- If all scores are close to 0.5 than the entire sample does not seem to have clearly distinct anomalies

Python example

Okay, so now let's see a hand-on example. For simplicity I will work on an artificial, 2-dimensional dataset. This way we can monitor the outlier identification process on a plot.

First, I need to generate observations. I will start with observations that will be considered normal and will be used to train the model (training and scoring in Python's scikit-learn implementation of Isolation Forest are analogous to all other machine learning algorithms). The second group are new observations, coming from the same distribution as the training ones. Lastly, I generate the outliers.

```
# importing libaries ----
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import savefig
from sklearn.ensemble import IsolationForest
```

```
# Generating data ----

rng = np.random.RandomState(42)

# Generating training data
X_train = 0.2 * rng.randn(1000, 2)
X_train = np.r_[X_train + 3, X_train]
X_train = pd.DataFrame(X_train, columns = ['x1', 'x2'])

# Generating new, 'normal' observation
X_test = 0.2 * rng.randn(200, 2)
X_test = np.r_[X_test + 3, X_test]
X_test = pd.DataFrame(X_test, columns = ['x1', 'x2'])

# Generating outliers
```

```
X_outliers = rng.uniform(low=-1, high=5, size=(50, 2))
X_outliers = pd.DataFrame(X_outliers, columns = ['x1',
'x2'])
```

Figure 2 presents the generated dataset. As desired, training and 'normal' observations are basically stacked on each other, while outliers are spread over. Due to random nature of the outliers some of them are overlapping with the training/normal observations, but I will account for that later.

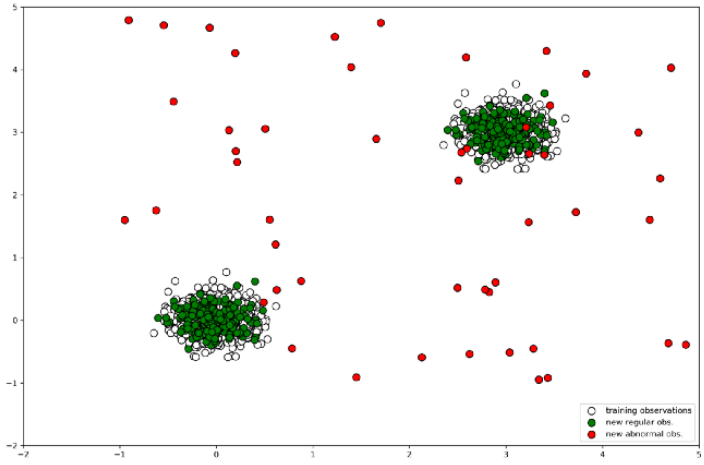

Figure 2 Generated Dataset

Now I need to train the Isolation Forest on the training set. I am using the default settings here. One thing worth noting is the contamination parameter, which specifies the percentage of observations we believe to be outliers (scikit-learn's default value is 0.1).

```
# Isolation Forest ----

# training the model
clf = IsolationForest(max_samples=100, random_state=rng)
clf.fit(X_train)

# predictions
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)
```

Okay, so now we have the predictions. How to assess the performance? We know that the test set contains only observations from the same

distribution as the normal observations. So, all of the test set observations should be classified as normal. And vice versa for the outlier set. Let's look at the accuracy.

```
# new, 'normal' observations ----
print("Accuracy:",
list(y_pred_test).count(1)/y_pred_test.shape[0])
# Accuracy: 0.93
```

```
# outliers ----
print("Accuracy:",
list(y_pred_outliers).count(-1)/y_pred_outliers.shape[0])
# Accuracy: 0.96
```

At first this looks pretty good, especially considering the default settings, however, there is one issue still to consider. As the outlier data was generated randomly, some of the outliers are actually located within the normal observations. To inspect it more carefully, I will plot the normal observation dataset together with labeled outlier set. We can see that some of the outliers lying within the normal observation sets were correctly classified as regular observations, with a few of them being misclassified. What we could do is to try different parameter specifications (contamination, number of estimators, number of samples to draw for trining the base estimators etc.) to get a better fit. But for now, these results are satisfactory.

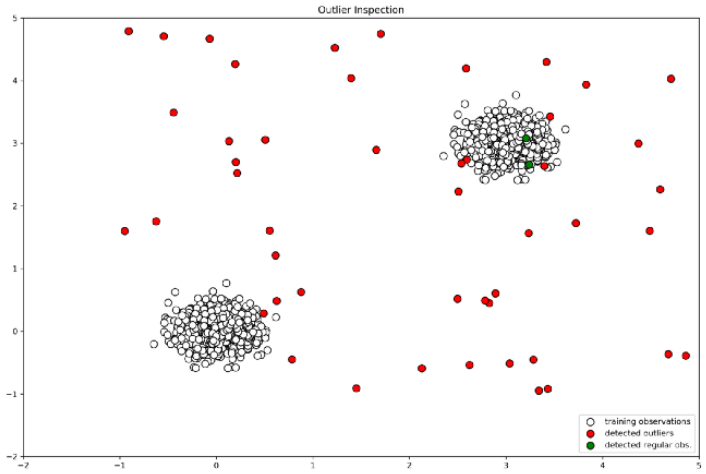

Figure 3 Inspecting outlier classification

Summing up:

- Isolation Forest is an outlier detection technique that identifies anomalies instead of normal observations
- Similarly to Random Forest it is built on an ensemble of binary (isolation) trees
- It can be scaled up to handle large, high-dimensional datasets

This was my first article here and in case I write some more I will try to improve the level of both writing and editing. Any constructive feedback is always welcome :)

Code used in this article can be found on my GitHub.

References:

[1] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on* (pp. 413–422). IEEE.

[2] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html