Data Science

Anomaly Detection Using Isolation Forest in Python

From bank fraud to preventative machine maintenance, anomaly detection is an incredibly useful and common application of machine learning.

4 years ago • 9 min read

By Dhiraj K

From bank fraud to preventative machine maintenance, anomaly detection is an incredibly useful and common application of machine learning. The isolation forest algorithm is a simple yet powerful choice to accomplish this task.

In this article we'll cover:

1. An Introduction to Anomaly Detection
2. Use Cases of Anomaly Detection
3. What Is Isolation Forest?
4. Using Isolation Forest for Anomaly Detection
5. Implementation in Python

You can run the code for this tutorial for free on the ML Showcase.

So, let's get started!

Introduction to Anomaly Detection

An **outlier** is nothing but a data point that differs significantly from other data points in the given dataset.

Anomaly detection is the process of finding the outliers in the data, i.e. points that are significantly different from the majority of the other data points.

Large, real-world datasets may have very complicated patterns that are difficult to detect by just looking at the data. That's why the study of anomaly detection is an extremely important application of Machine Learning.

In this article we are going to implement anomaly detection using the isolation forest algorithm. We have a simple dataset of salaries, where a few of the salaries are anomalous. Our goal is to find those salaries. You could imagine this being a situation where certain employees in a company are making an unusually large sum of money, which might be an indicator of unethical activity.

Photo by Will Myers / Unsplash

Before we proceed with the implementation, let's discuss some of the use cases of anomaly detection.

Anomaly Detection Use Cases

Anomaly detection has wide applications across industries. Below are some of the popular use cases:

Banking. Finding abnormally high deposits. Every account holder generally has certain patterns of depositing money into their account. If there is an outlier to this pattern the bank needs to be able to detect and analyze it, e.g. for money laundering.

Finance. Finding the pattern of fraudulent purchases. Every person generally has certain patterns of purchases which they make. If there is an outlier to this pattern the bank needs to detect it in order to analyze it for potential fraud.

Healthcare. Detecting fraudulent insurance claims and payments.

Manufacturing. Abnormal machine behavior can be monitored for cost control. Many companies continuously monitor the input and output parameters of the machines they own. It is a well-known fact that before failure a machine shows abnormal behaviors in terms of these input or output parameters. A machine needs to be constantly monitored for anomalous behavior from the perspective of preventive maintenance.

Networking. Detecting intrusion into networks. Any network exposed to the outside world faces this threat. Intrusions can be detected early on using monitoring for anomalous activity in the network.

Now let's understand what the isolation forest algorithm in machine learning is.

What Is Isolation Forest?

Isolation forest is a machine learning algorithm for anomaly detection.
It's an unsupervised learning algorithm that identifies anomaly by isolating outliers in the data.

Isolation Forest is based on the Decision Tree algorithm. It isolates the outliers by randomly selecting a feature from the given set of features and then randomly selecting a split value between the max and min values of that feature. This random partitioning of features will produce shorter paths in trees for the anomalous data points, thus distinguishing them from the rest of the data.

In general the first step to anomaly detection is to construct a profile of what's "normal", and then report anything that cannot be considered normal as anomalous. However, the isolation forest algorithm does not work on this principle; it does not first define "normal" behavior, and it does not calculate point-based distances.

As you might expect from the name, Isolation Forest instead works by isolating anomalies explicitly isolating anomalous points in the dataset.

The Isolation Forest algorithm is based on the principle that anomalies are observations that are few and different, which should make them easier to identify. Isolation Forest uses an ensemble of Isolation Trees for the given data points to isolate anomalies.

Isolation Forest recursively generates partitions on the dataset by randomly selecting a feature and then randomly selecting a split value for the feature. Presumably the anomalies need fewer random partitions to be isolated compared to "normal" points in the dataset, so the anomalies will be the points which have a smaller path length in the tree, path length being the number of edges traversed from the root node.

Using Isolation Forest, we can not only detect anomalies faster but we also require less memory compared to other algorithms.

Isolation Forest isolates anomalies in the data points instead of profiling normal data points. As anomalies data points mostly have a lot shorter tree paths than the normal data points, trees in the isolation forest does not need to have a large depth so a smaller max_depth can be used resulting in low memory requirement.

This algorithm works very well with a small data set as well.

Let's do some exploratory data analysis now to get some idea about the given data.

Exploratory Data Analysis

Let's import the required libraries first. We are importing `numpy`, `pandas`, `seaborn` and `matplotlib`. Apart form that we also need to import IsolationForest from `sklearn.ensemble`.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
```

COPY

Once the libraries are imported we need to read the data from the csv to the pandas data frame and check the first 10 rows of data.

The data is a collection of salaries, in USD per year, of different professionals. This data has few anomalies (like salary too high or too low) which we will be detecting.

```
df = pd.read_csv('salary.csv')
df.head(10)
```

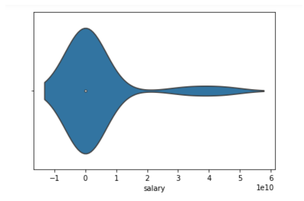
COPY

	salary
0	26100
1	73188
2	33333333333
3	14623
4	44444444444
5	95759
6	22368
7	90715
8	18507
9	11559

Dataset head

To get more of an idea of the data we have plotted a violin plot of salary data as shown below. A violin plot is a method of plotting numeric data.

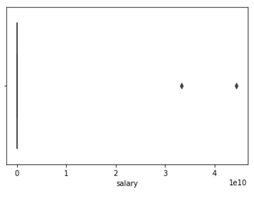
Typically a violin plot includes all the data that is in a box plot, a marker for the median of the data, a box or marker indicating the interquartile range, and possibly all sample points, if the number of samples is not too high.



Violin Plot for Salary

To get a better idea of outliers we may like to look at a box plot as well. This is also known as box-and-whisker plot. The box in box plot shows the quartiles of the dataset, while the whiskers shows the rest of the distribution.

Whiskers do not show the points that are determined to be outliers.
Outliers are detected by a method which is a function of the interquartile range.
In statistics the interquartile range, also known as mid spread or middle 50%, is a measure of statistical dispersion, which is equal to the difference between 75th and 25th percentiles.



Box Plot for Salary Indicating the Two outliers in the right

Once we have completed our exploratory data analysis, it's time to define and fit the model.

Define and Fit Model

We'll create a `model` variable and instantiate the IsolationForest class. We are passing the values of four parameters to the Isolation Forest method, listed below.

Number of estimators: `n_estimators` refers to the number of base estimators or trees in the ensemble, i.e. the number of trees that will get built in the forest. This is an integer parameter and is optional. The default value is 100.

Max samples: `max_samples` is the number of samples to be drawn to train each base estimator. If max_samples is more than the number of samples provided, all samples will be used for all trees. The default value of max_samples is 'auto'. If 'auto', then max_samples=min(256, n_samples)

Contamination: This is a parameter that the algorithm is quite sensitive to; it refers to the expected proportion of outliers in the data set. This is used when fitting to define the threshold on the scores of the samples. The default value is 'auto'. If 'auto', the threshold value will be determined as in the original paper of Isolation Forest.

Max features: All the base estimators are not trained with all the features available in the dataset. It is the number of features to draw from the total features to train each base estimator or tree.The default value of max features is one.

```
model=IsolationForest(n_estimators=50, max_samples='auto', contamination=float(0.1),max_features=1.0)
model.fit(df[['salary']])
```

COPY

```
IsolationForest(behaviour='deprecated', bootstrap=False, contamination=0.1,
                max_features=1.0, max_samples='auto', n_estimators=50,
                n_jobs=None, random_state=None, verbose=0, warm_start=False)
```

Isolation Forest Model Training Output

After we defined the model above we need to train the model using the data given. For this we are using the `fit()` method as shown above. This method is passed one parameter, which is our data of interest (in this case, the salary column of the dataset).

Once the model is trained properly it will output the IsolationForest instance as shown in the output of the cell above.

Now this is the time to add the scores and anomaly column of the dataset.

Add Scores and Anomaly Column

After the model is defined and fit, let's find the scores and anomaly column. We can find out the values of scores column by calling `decision_function()` of the trained model and passing the salary as parameter.

Similarly we can find the values of anomaly column by calling the `predict()` function of the trained model and passing the salary as parameter.

These columns are going to be added to the data frame `df`. After adding these two columns let's check the data frame. As expected, the data frame has three columns now: salary, scores and anomaly. A negative score value and a -1 for the value of anomaly columns indicate the presence of anomaly. A value of 1 for the anomaly represents the normal data.

Each data point in the train set is assigned an anomaly score by this algorithm. We can define a threshold, and using the anomaly score, it may be possible to mark a data point as anomalous if its score is greater than the predefined threshold.

```
df['scores']=model.decision_function(df[['salary']])
df['anomaly']=model.predict(df[['salary']])
df.head(20)
```

COPY

	salary	scores	anomaly
0	26100	0.121556	1
1	73188	0.063679	1
2	33333333333	-0.186302	-1
3	14623	0.144374	1
4	44444444444	-0.211409	-1
5	95759	0.113981	1
6	22368	0.147079	1
7	90715	0.020700	1
8	18507	0.119701	1
9	11559	0.104399	1
10	26800	0.146092	1

Added scores and Anomaly Column for Isolation Forest

After adding the scores and anomalies for all the rows in the data, we will print the predicted anomalies.

Print Anomalies

To print the predicted anomalies in the data we need to analyse the data after addition of scores and anomaly column. As you can see above for the predicted anomalies the anomaly column values would be -1 and their scores will be negative.

Using this information we can print the predicted anomaly (two data points in this case) as below.

```
anomaly=df.loc[df['anomaly']==-1]
anomaly_index=list(anomaly.index)
print(anomaly)
```

COPY

```
        salary    scores  anomaly
2  33333333333 -0.194531       -1
4  44444444444 -0.215396       -1
```

Anomaly output

Note that we could print not only the anomalous values but also their index in the dataset, which is useful information for further processing.

Evaluating the model

For evaluating the model let's set a threshold as salary > 99999 is an outlier.Let us find out the number of outlier present in the data as per the above rule using code as below.

```
outliers_counter = len(df[df['salary'] > 99999])
outliers_counter
```

COPY

outlier_counter = 2

Let us calculate the accuracy of the model by finding how many outlier the model found divided by how many outliers present in the data.

```
print("Accuracy percentage:", 100*list(df['anomaly']).count(-1)/(outliers_counter))
```

COPY

Accuracy percentage: 100 %

End Notes

In this tutorial we learned what anomalies are, and how the Isolation Forest algorithm can be used to detect them. We also discussed various exploratory data analysis graphs like violin plot and box plot for this problem.

Finally we implemented the Isolation Forest Algorithm and printed the real outliers in the data.

I hope you liked the article and you may like to use it in your project in future if required.

Happy Learning!

- Tags:
- Data Science
- Machine Learning

Spread the word

- Share
- Tweet
- Share
- Copy
- Email

https://blog.paperspace.com/

public
Next article

Cerebral Sandwiches and Connectomics of the Brain

public
Previous article

How AI And Deep Learning Are Shaping Education: An Interview With Terry Sejnowski

Keep reading

public

Encoding Categorical Data with One-hot Encoding

7 days ago • 8 min read
public

XGBoost: A Comprehensive Guide, Model Overview, Analysis, and Code Demo using Paperspace GPUs

2 months ago • 20 min read
public

Problem-Solving with Synthetic Data: A Discussion on Practical Applications

3 months ago • 5 min read