# OUTLIER AND ANOMALY DETECTION WITH MACHINE LEARNING

In this article, we'll explain how to do outlier detection. Outlier detection is important in a variety of scenarios: credit card fraud, cyber security, and to detect, for example, faulty equipment in systems.

*(This article is part of our scikit-learn Guide. Use the right-hand menu to navigate.)*

## The approach

The simplest approach for outlier detection is to assume a normal distribution and then set a threshold at some number of standard deviations. That's called the **z-score**.

In this example, we're using a different approach—**an isolation forest**. It's a type of decision tree that picks values at random, splits each data point between a randomly selected range, then picks another randomly picked value on an ever-decreasing range of values until it runs out of values to split. Lastly, it flags as outliers those that are on the shortest path in that tree. This flags outliers by calculation an **anomaly score**.

In the sample below we mock sample data to illustrate how to do anomaly detection using an isolation forest within the scikit-learn machine learning framework.

## The code, explained

The code for this example is here.

First, we pull 100 samples from a standard normal distribution with mean 4 and standard deviation 1 to create a 100x2 matrix:

```
train = rng.normal(4,1,(100,2))
```

Now, train that model:

```
from sklearn.ensemble import IsolationForest

clf = IsolationForest(behaviour='new', max_samples=100,
                      random_state=rng, contamination='auto')

clf.fit(train)
```

Next, we create another sample.

```
typical =  rng.normal(4,1,(100,2))
```

If we print that we see that the values range between 2 and 6. So, we create outliers of values 8 and 1:

```
outliers = np.array(,
])
```

Here is the complete code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(20)

train = rng.normal(4,1,(100,2))


typical =  rng.normal(4,1,(100,2))

outliers = np.array(,
])

clf = IsolationForest(behaviour='new', max_samples=100,
                      random_state=rng, contamination='auto')

clf.fit(train)

pred_train = clf.predict(train)
pred_test = clf.predict(typical)
pred_outliers = clf.predict(outliers)
```

Here is a sample of the data. It ranges between above 1 and below 6, so we will mock data above and below 1 and 6, which the algorithm should flag as outliers.

```
typical
```

```
array(,
       ,
       ,
       ,
       ,
       ,
       ,
```

Then we run three predictions and plot them, plotting the second column in each matrix on the x columns and the first column on the y axis , just as an easy way to visualize the pairs. Since they are drawn from a normal distribution, they cluster around the middle.

```
pred_train = clf.predict(train)
pred_test = clf.predict(typical)
pred_outliers = clf.predict(outliers)
```

```
fig, axis = plt.subplots(2)

axis.scatter(train, train )

axis.scatter(typical, typical  , c='g'  )
axis.scatter(outliers, outliers , c='r'  )
```

The points and are outliers which is easy to see, since we plot them in red.