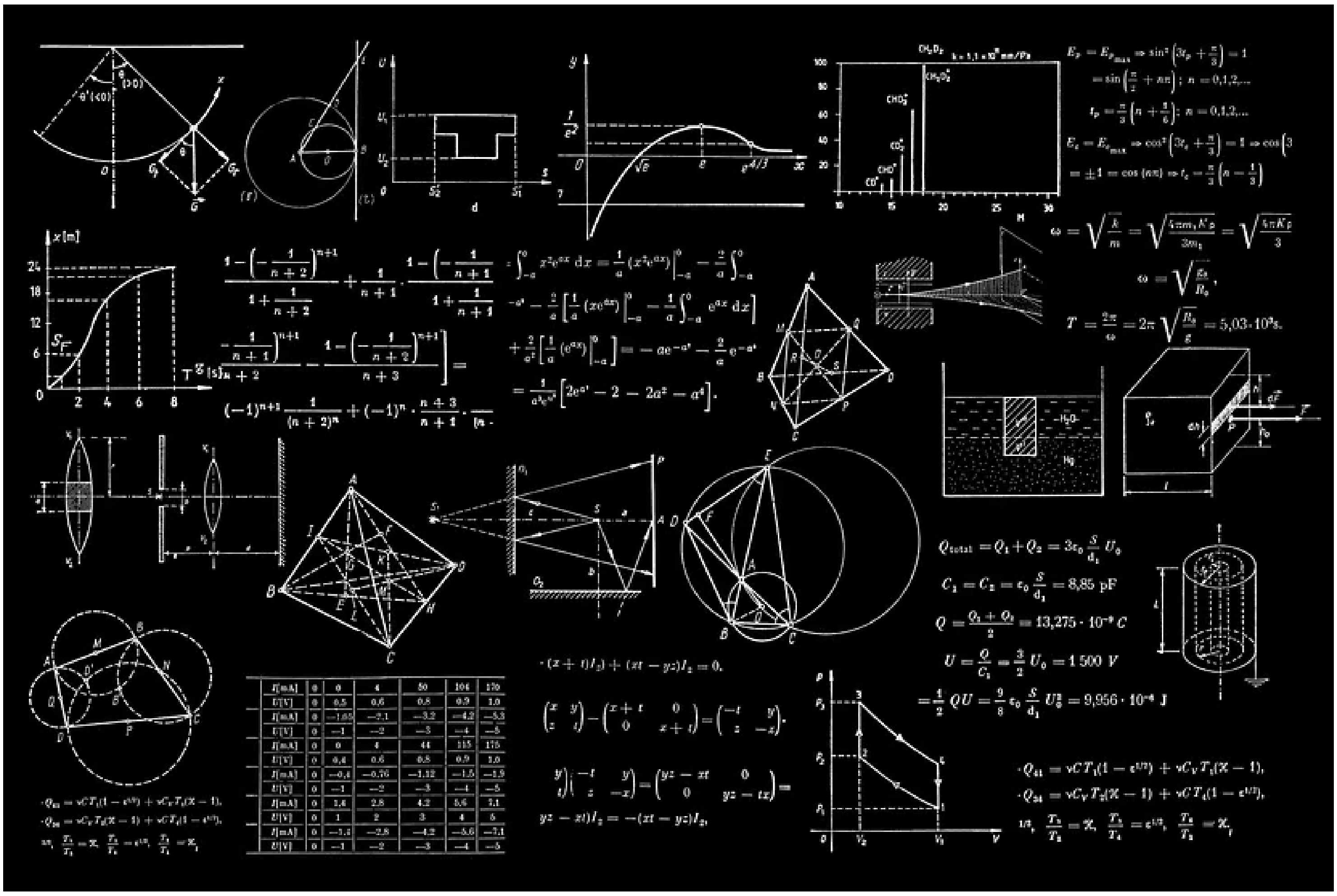


Photo by Dan Cristian Pădureț on Unsplash

Isolation Forest

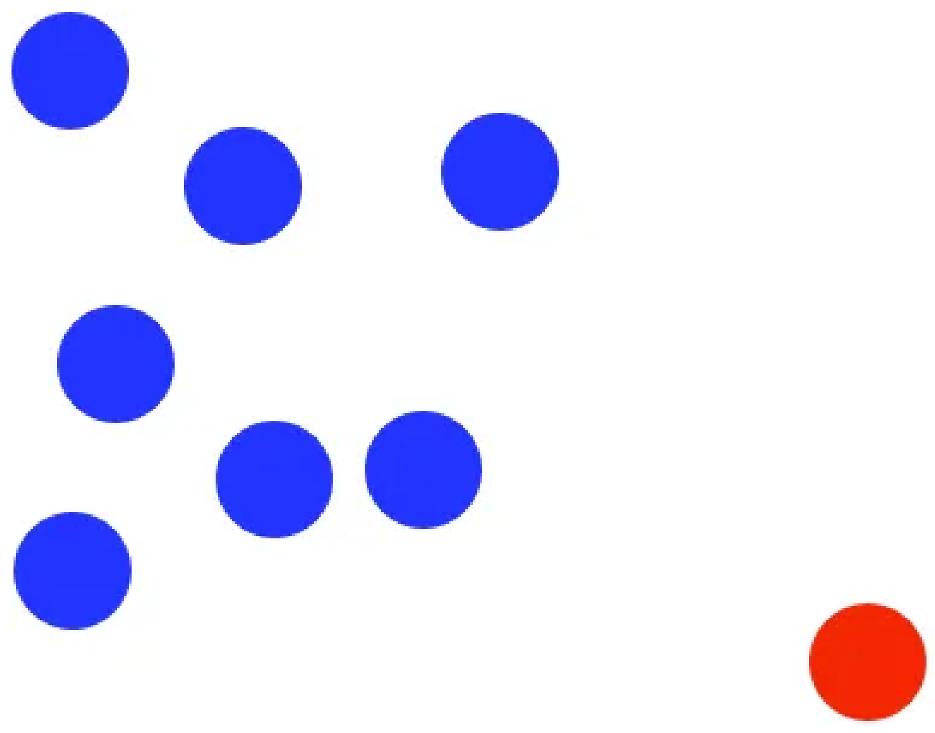
Cory Maklin · Follow

5 min read · Jul 15, 2022

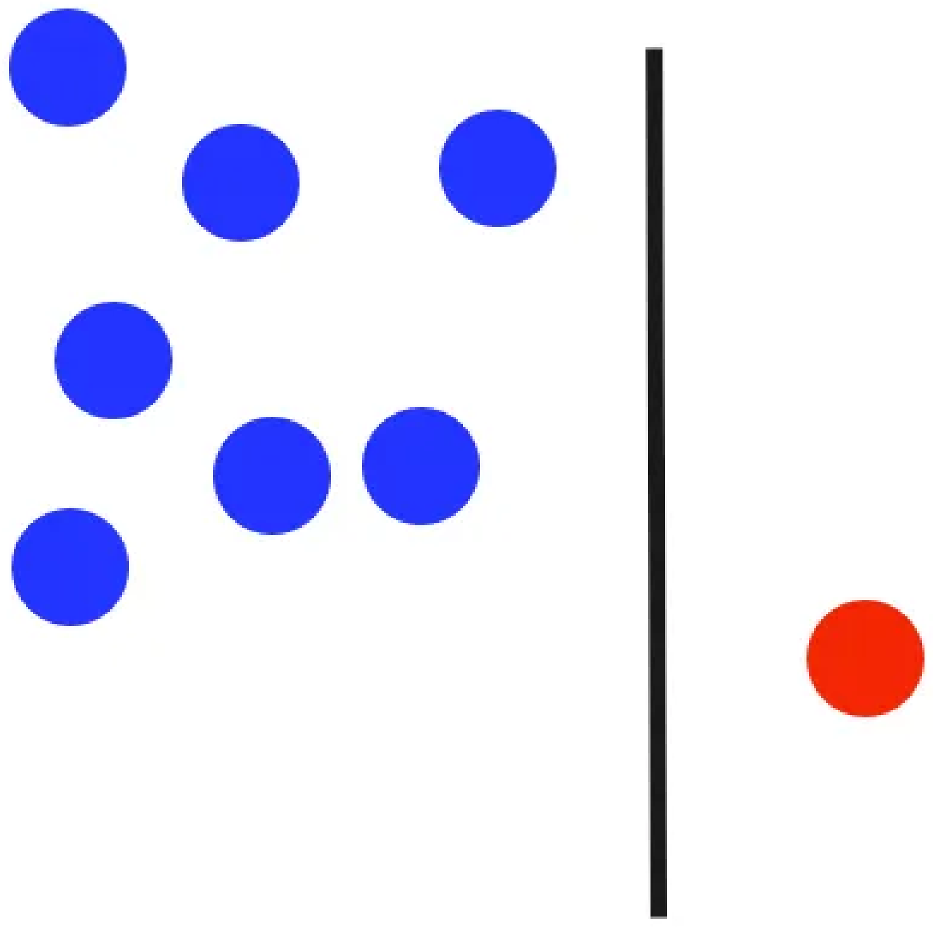
Isolation Forest is an unsupervised machine learning algorithm for anomaly detection. As the name implies, Isolation Forest is an ensemble method (similar to random forest). In other words, it use the average of the predictions by several decision trees when assigning the final anomaly score to a given data point. Unlike other anomaly detection algorithms, which first define what's "normal" and then report anything else as anomalous, Isolation Forest attempts to isolate anomalous data points from the get go.

Algorithm

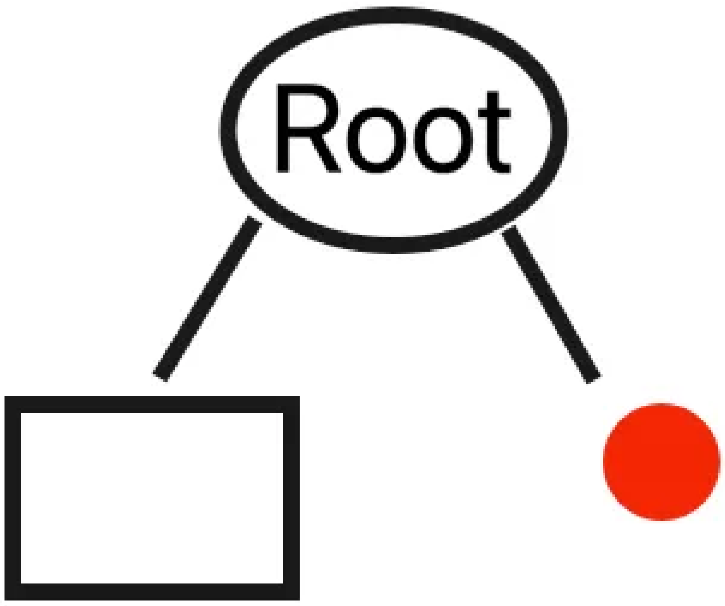
Suppose we had the following data points:



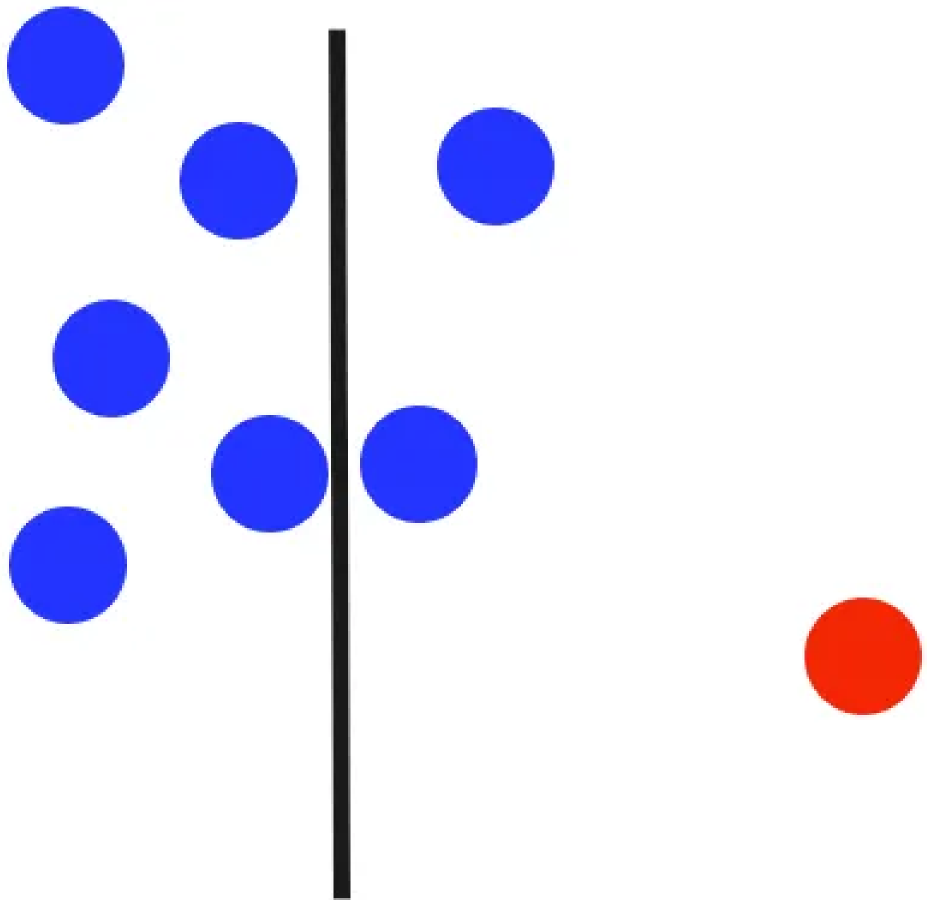
The isolation forest algorithm selects a random dimension (in this case, the dimension associated with the x axis) and randomly splits the data along that dimension.



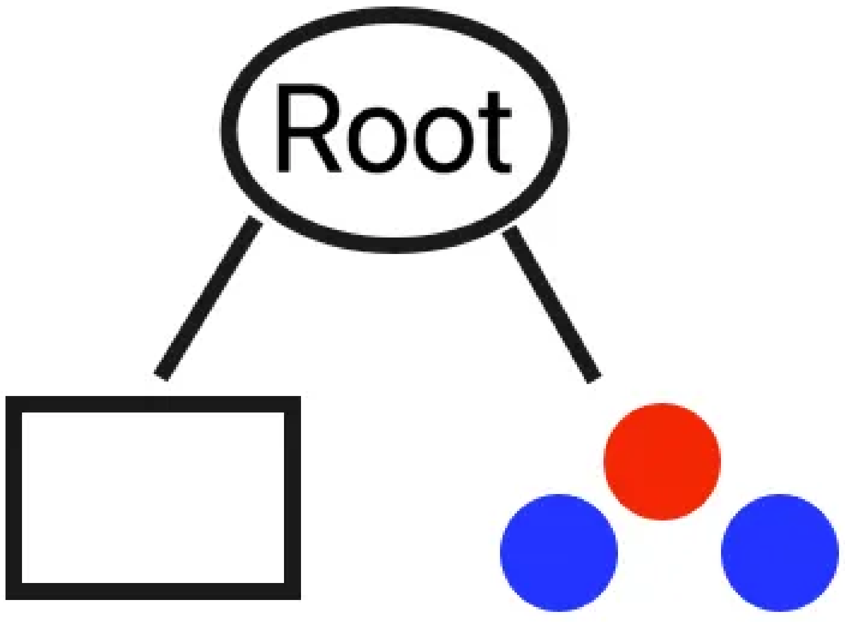
The two resulting subspaces define their own sub tree. In this example, the cut happens to separate a lone point from the remainder of the dataset. The first level of the resulting binary tree consists of two nodes, one which will consist of the subtree of points to the left of the initial cut and the other representing the single point on the right.



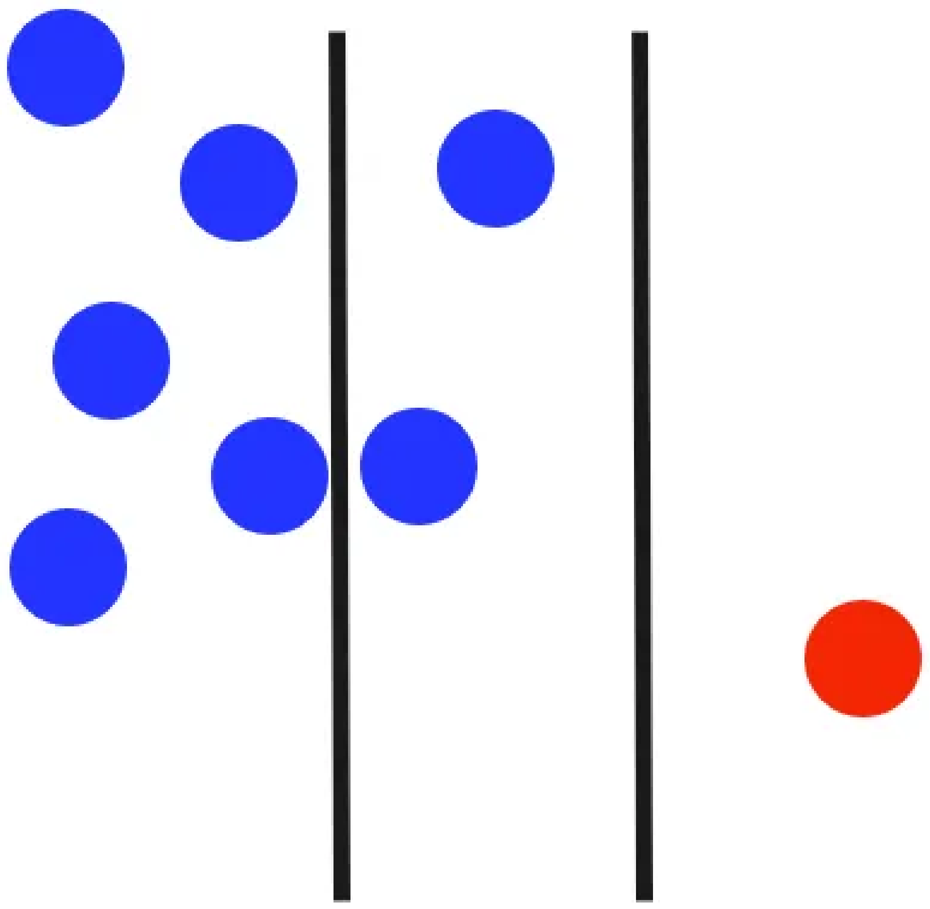
It’s important to note, the other trees in the ensemble will select different starting splits. In the following example, the first split doesn’t isolate the outlier.



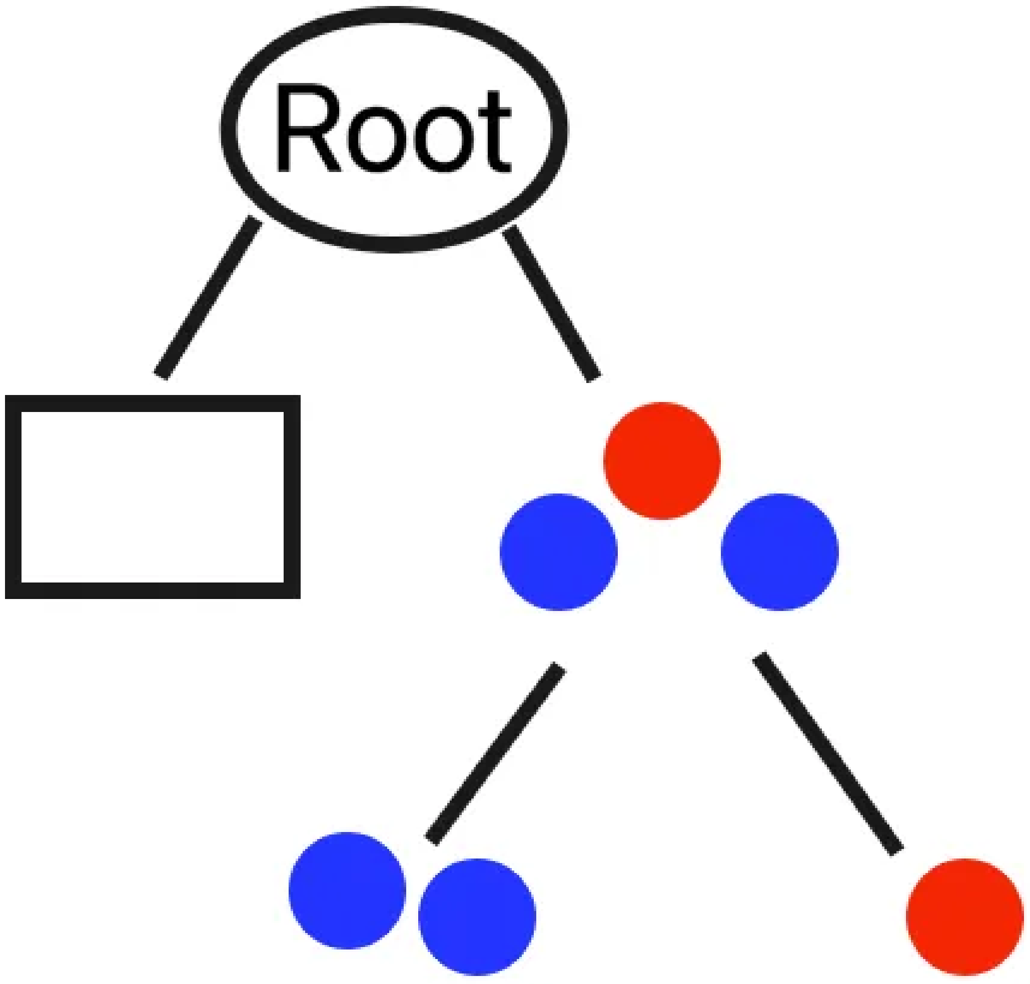
We end up with a tree consisting of two nodes, one that contains the points to the left of the line and the other representing the points on the right side of the line.

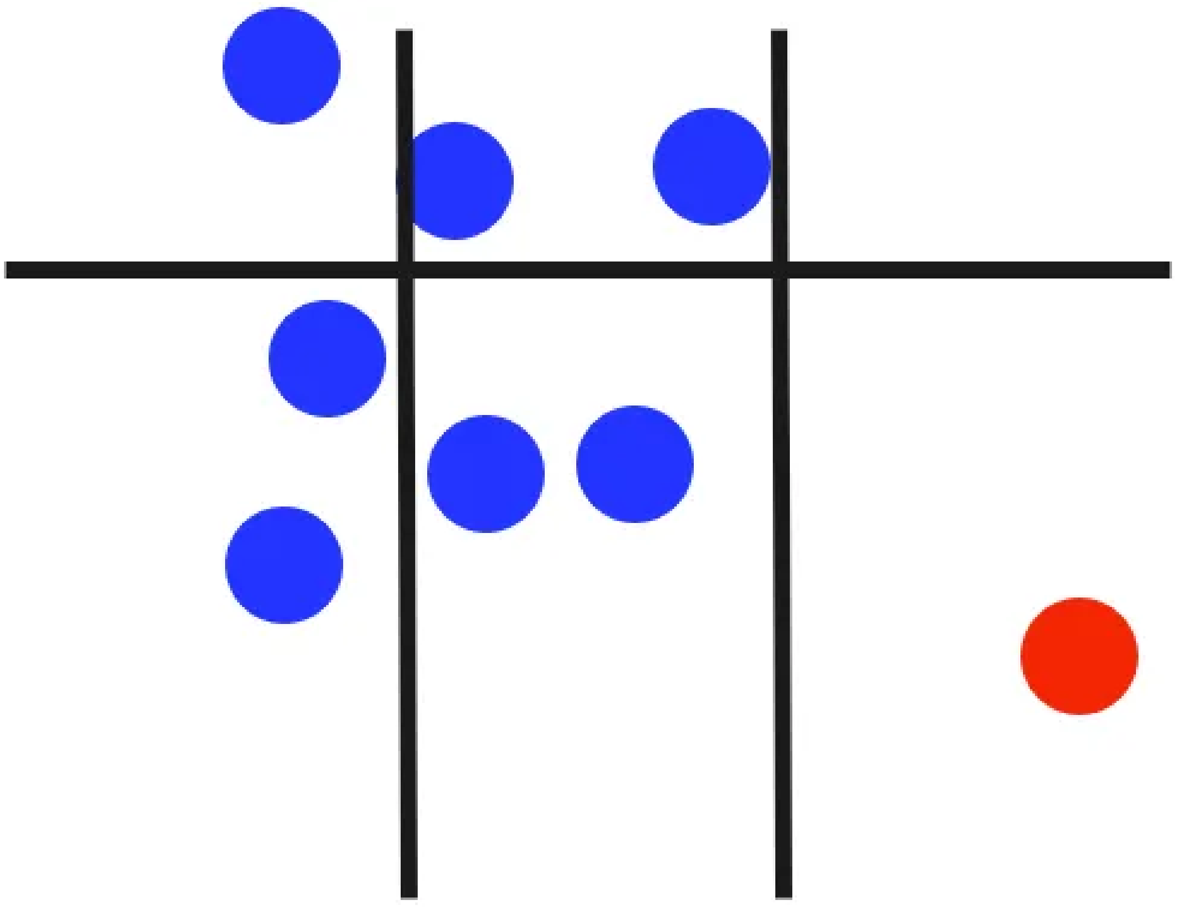


The process is repeated until every leaf of the tree represents a single data point from the dataset. In our example, the second iteration manages to isolate the outlier.

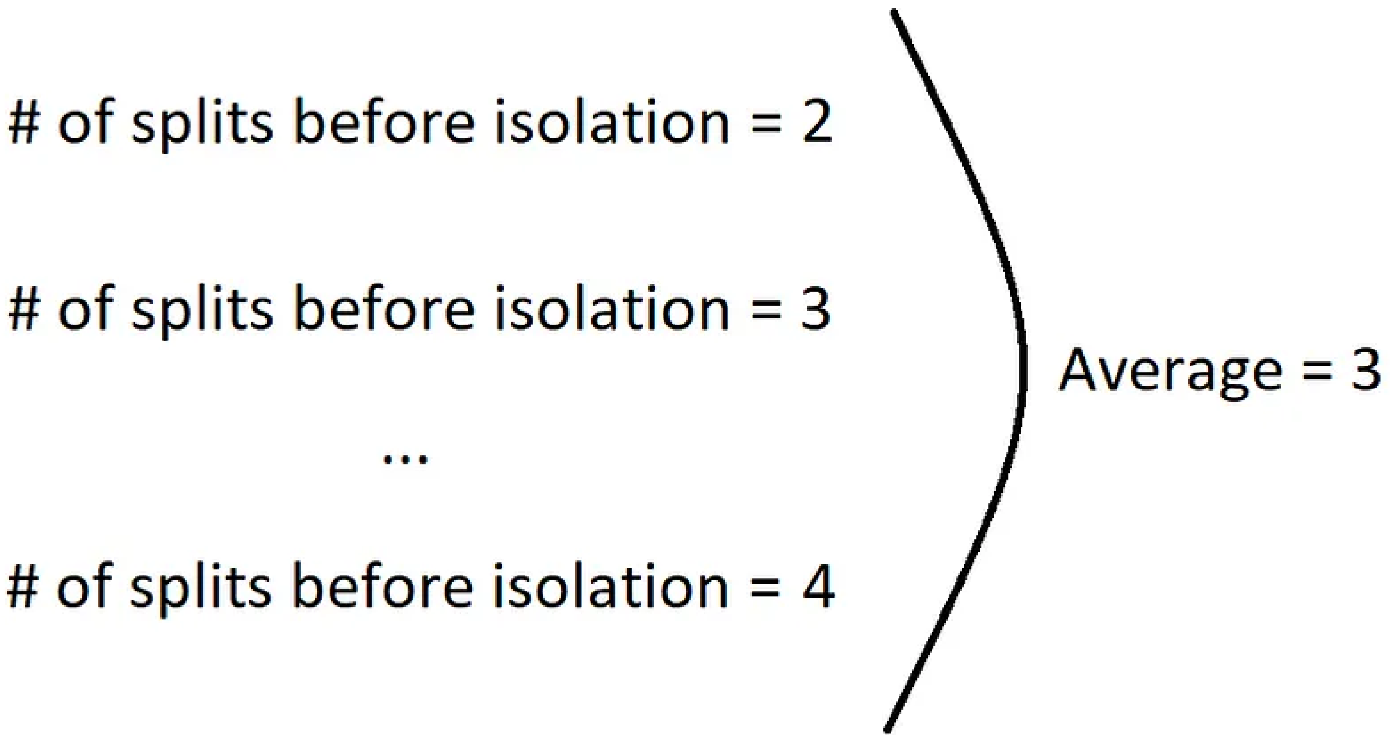


After this step, the tree would look as follows:


Root

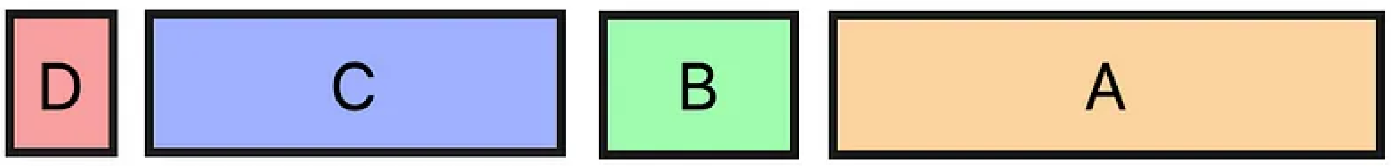


On average, an anomalous data point is going to be isolated in a bounding box at a smaller tree depth than other points. When performing inference using a trained Isolation Forest model the final anomaly score is reported as the average across scores reported by each individual decision tree.

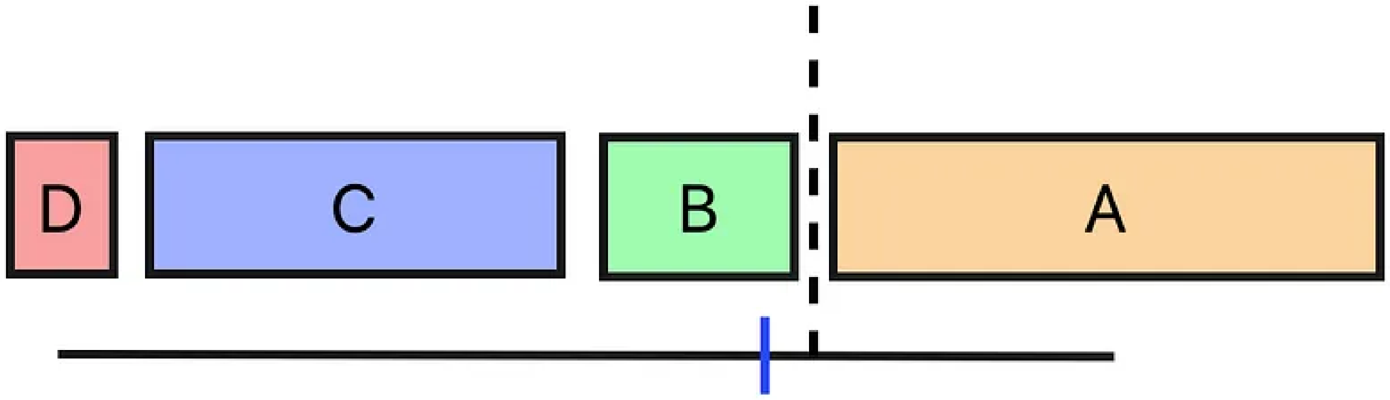

# of splits before isolation = 2
# of splits before isolation = 3
...
# of splits before isolation = 4
Average = 3

Categorical Variables

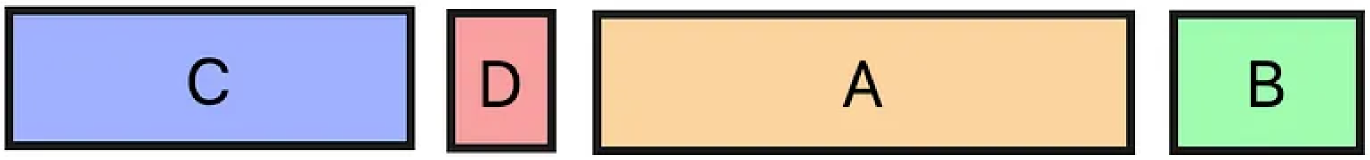
If you're like me, you're probably asking yourself how this would work with categorical variables. Assuming that a value that is less observed is anomalous, the Isolation Forest algorithm can make use of categorical variables by representing them as rectangles where the size of rectangle is proportional to the frequency of occurrence.



We consider the set of possible values between the middle of the first value and the middle of the last value. We select a random point along the domain then determine the closest edge of a given rectangle. This is used for our split.



To ensure fairness, the other trees in the forest will use a different ordering.



Python

To start, import the following libraries:

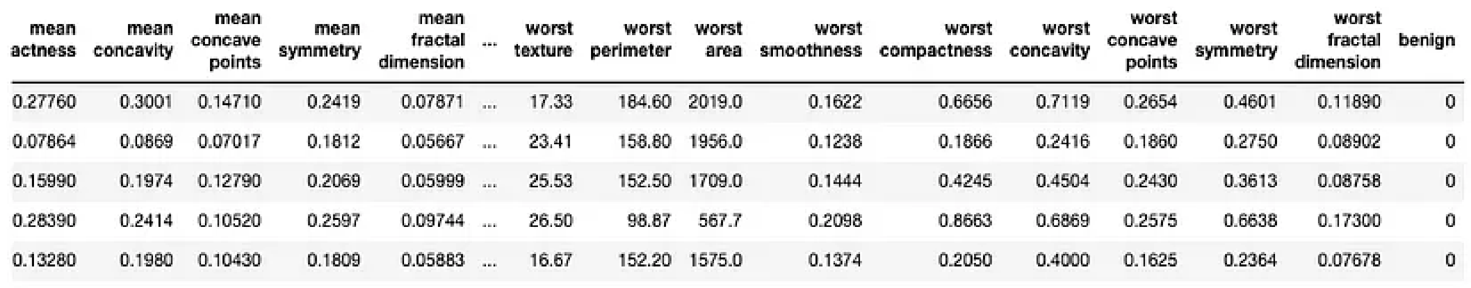
```
from sklearn.ensemble import IsolationForest
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils import resample
import pandas as pd
```

In the proceeding tutorial, we'll be working with the breast cancer dataset from the UCI machine learning repository. Fortunately, the `scitkit-learn` library provides a wrapper function for downloading the data.

```
breast_cancer = load_breast_cancer()
df = pd.DataFrame(data=breast_cancer.data,
columns=breast_cancer.feature_names)
df["benign"] = breast_cancer.target
```

As we can see, the dataset contains 30 numerical features and a target value of 0 and 1 for benign and malignant tumors, respectively.

```
df.head()
```



mean actness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	benign
0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

For our use case, we will assume that a malignant label is anomalous. The dataset contains a relatively high number of malignant tumors. Thus, we make use of downsampling.

```
majority_df = df[df["benign"] == 1]
minority_df = df[df["benign"] == 0]
minority_downsampled_df = resample(minority_df, replace=True,
n_samples=30, random_state=42)
downsampled_df = pd.concat([majority_df,
minority_downsampled_df])
```

After downsampling, there are over 10x more samples of the majority class than the minority class.

```
downsampled_df["benign"].value_counts()
```

```
1    357
0     30
Name: benign, dtype: int64
```

We save the features and target as separate variables.

```
y = downsampled_df["benign"]
X = downsampled_df.drop("benign", axis=1)
```

We set a portion of the total data aside for testing.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
```

Next, we create an instance of the `IsolationForest` class.

```
model = IsolationForest(random_state=42)
```

We train the model.

```
model.fit(X_train, y_train)
```

We predict the data in the test set.

```
y_pred = model.predict(X_test)
```

The `IsolationForest` assigns a value of `-1` instead of 0. Therefore, we replace it to ensure we only have 2 distinct values in our confusion matrix.

```
y_pred[y_pred == -1] = 0
```

As we can see, the algorithm does a good job of predicting what data points are anomalous.

```
confusion_matrix(y_test, y_pred)

array([[ 7,  2],
       [ 5, 83]])
```

Machine Learning
Data Science
Artificial Intelligence
Math
Programming