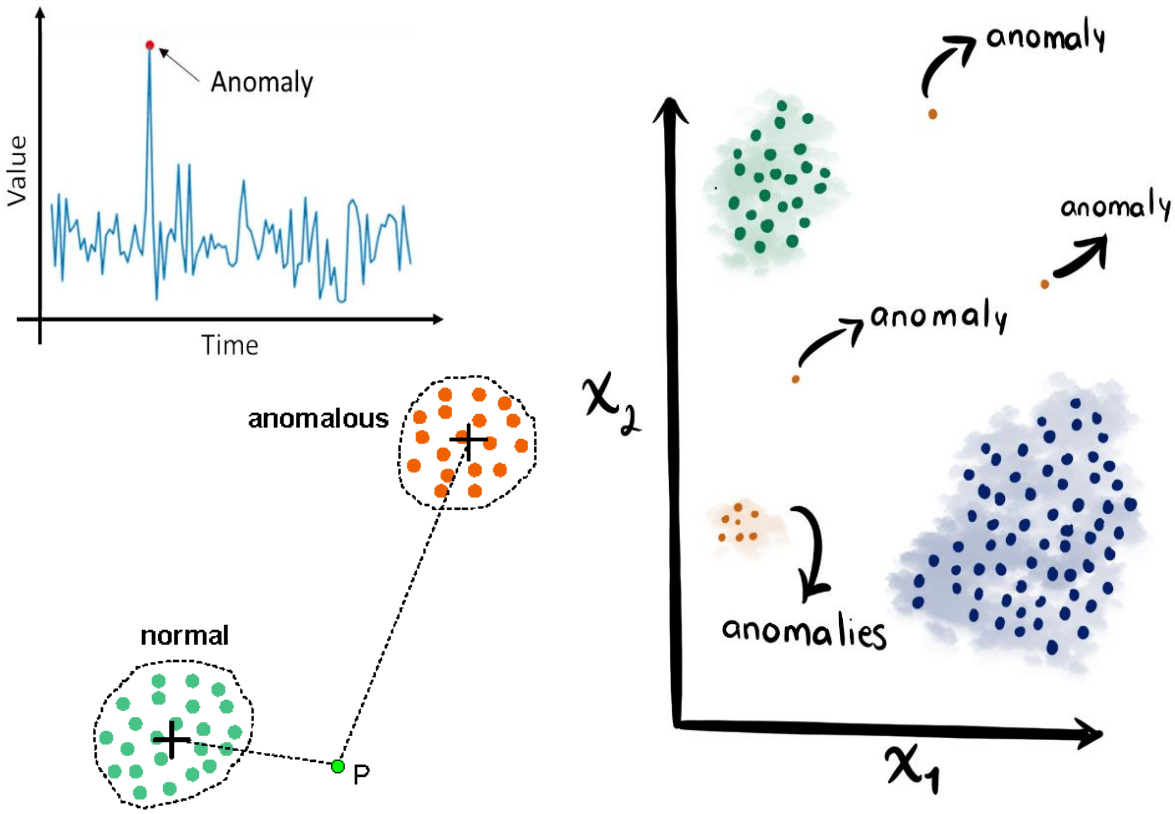
Pratistha Gaur

Anomaly Detection using Isolation Forest Algorithm
In a Synthetic Dataset with Gaussian Inliers and Uniformly Distributed Outliers

Anomaly Detection is a crucial task in many domains, including fraud detection, intrusion detection, and system monitoring.
One popular algorithm for anomaly detection is the **IsolationForest** algorithm. This algorithm constructs random decision trees and isolates the outliers in the dataset. The algorithm is easy to use, has low computational complexity, and is well suited for high-dimensional datasets.

This code demonstrates how to use the Isolation Forest algorithm for anomaly detection on a synthetic dataset. The dataset contains two clusters of Gaussian distributed points and some uniformly distributed outliers. The script generates the dataset using NumPy, splits it into a training and testing set using the **train_test_split** function from scikit-learn, and trains an IsolationForest model on the training set using the Isolation Forest class from scikit-learn. The **max_samples** parameter is set to 100, which specifies the number of samples to draw from the dataset to build each tree. The **random_state** parameter is set to 0 to ensure reproducibility.
The script then generates decision boundaries based on two different methods using the **DecisionBoundaryDisplay** class from **scikit-learn**. The binary decision function method generates a decision boundary that separates the inliers from the outliers based on the predicted class labels. The path length decision function method generates a decision boundary based on the average path length of the samples in the trees. Both methods can help visualize how the Isolation Forest algorithm detects the outliers.
The resulting plot shows the decision boundaries along with the true class labels of the data points. The plot reveals that the Isolation Forest algorithm can detect most of the outliers correctly, but some inliers are incorrectly labelled as outliers. This is because the algorithm's performance depends on the distribution of the inliers and outliers in the dataset. In some cases, other algorithms may be better suited for anomaly detection.
Overall, this code provides a good example of how to use the **IsolationForest algorithm** in Python for anomaly detection. The code can serve as a starting point for exploring the algorithm's performance on different types of datasets and for comparing it with other algorithms for anomaly detection.



Dataset description

Context:

The dataset used in this code is a synthetic dataset created using **NumPy**. It consists of two clusters of points, where each cluster is generated from a **two-dimensional Gaussian distribution**. One of the clusters is elongated and has a covariance matrix that is not diagonal, while the other is spherical and has a diagonal covariance matrix. The elongated cluster is centered at the point (2, 2), while the spherical cluster is centered at the point (-2, -2). The dataset also contains some uniformly distributed outliers that are generated using the uniform distribution function from NumPy. The number of samples in the **dataset is set to 120**, and the **number of outliers is set to 40**. The script concatenates the clusters and outliers to create a single dataset.
The resulting dataset has **160 samples** in total, where **80 samples belong to each cluster and 40 samples are outliers**.
The true class labels for the dataset are generated by assigning the **label +1 to the inliers** and the **label -1 to the outliers**. The resulting dataset is then split into a training set and a testing set using the **train_test_split function** from scikit-learn. The training set is used to train the **IsolationForest** model, and the testing set is used to evaluate the model's performance.

Purpose of choosing this dataset:

The purpose of using a synthetic dataset is to demonstrate the performance of the **IsolationForest** algorithm in detecting outliers in a controlled environment where the true class labels are known. **Synthetic datasets** are often used in machine learning research to **evaluate the performance of algorithms in different scenarios and to compare them against other algorithms**. Using a synthetic dataset also allows researchers to study the behavior of algorithms in controlled experiments and to draw conclusions about their strengths and weaknesses.

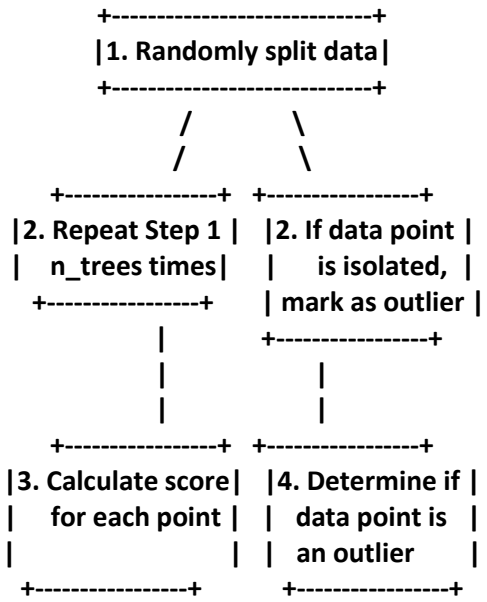
Design (Method and methodology)

Methodology:

The methodology used in this program is based on the IsolationForest algorithm, which is an unsupervised machine learning algorithm for outlier detection. The IsolationForest algorithm is based on the principle that outliers are points that are isolated from the rest of the dataset. The algorithm works by constructing an ensemble of binary decision trees, where each tree is trained on a random subset of the data. To determine whether a point is an outlier, the algorithm computes the average path length of the point in the trees of the forest. Points with a shorter average path length are considered more isolated and are therefore more likely to be outliers.
The program first generates a synthetic dataset using NumPy, consisting of two clusters of points and some uniformly distributed outliers. The dataset is then split into a training set and a testing set using **the train_test_split function** from scikit-learn. The IsolationForest algorithm is then applied to the training set using the IsolationForest class from scikit-learn. The algorithm is configured to use a maximum of 100 samples per tree and a random state of 0 for reproducibility. After training the model, the program uses the **DecisionBoundaryDisplay class** from scikit-learn to visualize the decision boundary of the model in two different ways.
The first visualization shows the binary decision boundary of the IsolationForest model based on the predicted class labels. The second visualization shows the decision boundary of the model based on the path lengths of the points in the trees of the forest.

The purpose of these visualizations is to help understand how the IsolationForest algorithm separates the inliers from the outliers in the dataset. Finally, the program evaluates the performance of the model on the testing set using various metrics, such as accuracy, precision, recall, and F1-score. The results of the evaluation can be used to determine the effectiveness of the IsolationForest algorithm in detecting outliers in the dataset.

Block Diagram:



Flowchart:

Start
|
|---Generate synthetic dataset using NumPy
|
|---Split dataset into training and testing sets using **train_test_split** function
|
|---Train an **IsolationForest model** on the training set using **IsolationForest class**
|
|---Visualize the decision boundary of the model using **DecisionBoundaryDisplay** class
| |---Visualize binary decision boundary of the model based on predicted class labels
| |---Visualize decision boundary of the model based on path lengths of points
|
|---Evaluate performance of the model on the testing set using various metrics
|
|---Visualize the dataset with true and predicted class labels using **matplotlib**
|
End

Domain: The code finds its application in detecting anomalies in network traffic, the domain of this project is thus, in the field of network security and anomaly detection.

Algorithm: The program uses the Isolation Forest algorithm for anomaly detection.

Technology: The following technologies and libraries are used in this project:

- Python: a popular high-level programming language used for various applications, including data science and machine learning.
- NumPy: a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- scikit-learn (sklearn): a Python library built on top of NumPy, providing a range of machine learning algorithms for classification, regression, and clustering tasks, among others.

- matplotlib: a plotting library for the Python programming language and its numerical mathematics extension NumPy.
- Jupyter Notebook: a web-based interactive computing environment for creating, running, and sharing Jupyter notebook documents containing live code, equations, visualizations, and narrative text.

Implementation

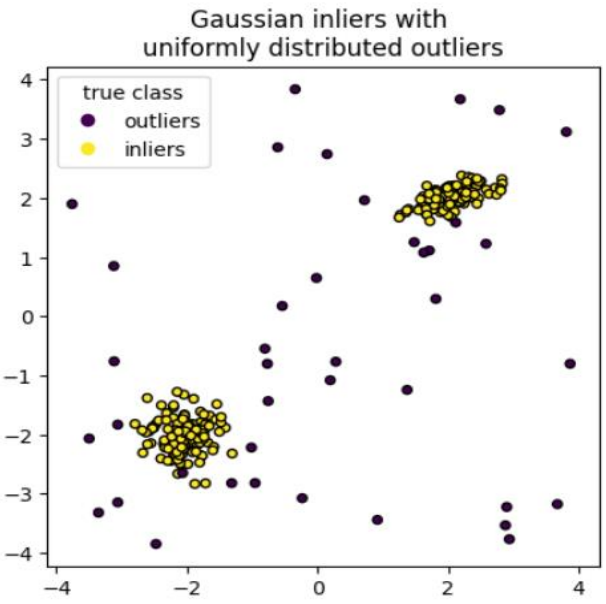
+ Code + Text

```
import numpy as np
from sklearn.model_selection import train_test_split

n_samples, n_outliers = 120, 40
rng = np.random.RandomState(0)
covariance = np.array([[0.5, -0.1], [0.7, 0.4]])
cluster_1 = 0.4 * rng.randn(n_samples, 2) @ covariance + np.array([2, 2])  # general
cluster_2 = 0.3 * rng.randn(n_samples, 2) + np.array([-2, -2])  # spherical
outliers = rng.uniform(low=-4, high=4, size=(n_outliers, 2))

X = np.concatenate([cluster_1, cluster_2, outliers])
y = np.concatenate(
    [np.ones((2 * n_samples), dtype=int), -np.ones((n_outliers), dtype=int)]
)

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

```
[ ] import matplotlib.pyplot as plt

scatter = plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor="k")
handles, labels = scatter.legend_elements()
plt.axis("square")
plt.legend(handles=handles, labels=["outliers", "inliers"], title="true class")
plt.title("Gaussian inliers with \nuniformly distributed outliers")
plt.show()
```

+ Code + Text

[ ]

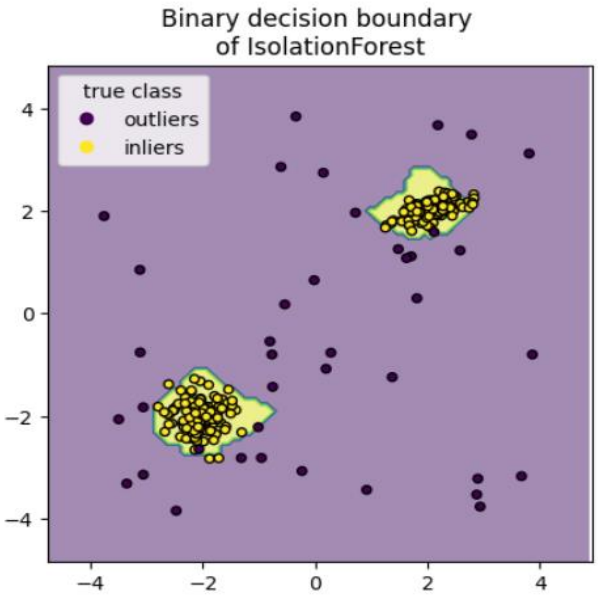


```
[ ] from sklearn.ensemble import IsolationForest

    clf = IsolationForest(max_samples=100, random_state=0)
    clf.fit(X_train)
```

```
IsolationForest(max_samples=100, random_state=0)
```

```
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay

disp = DecisionBoundaryDisplay.from_estimator(
    clf,
    X,
    response_method="predict",
    alpha=0.5,
)
disp.ax_.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor="k")
disp.ax_.set_title("Binary decision boundary \nof IsolationForest")
plt.axis("square")
plt.legend(handles=handles, labels=["outliers", "inliers"], title="true class")
plt.show()
```

+ Code + Text

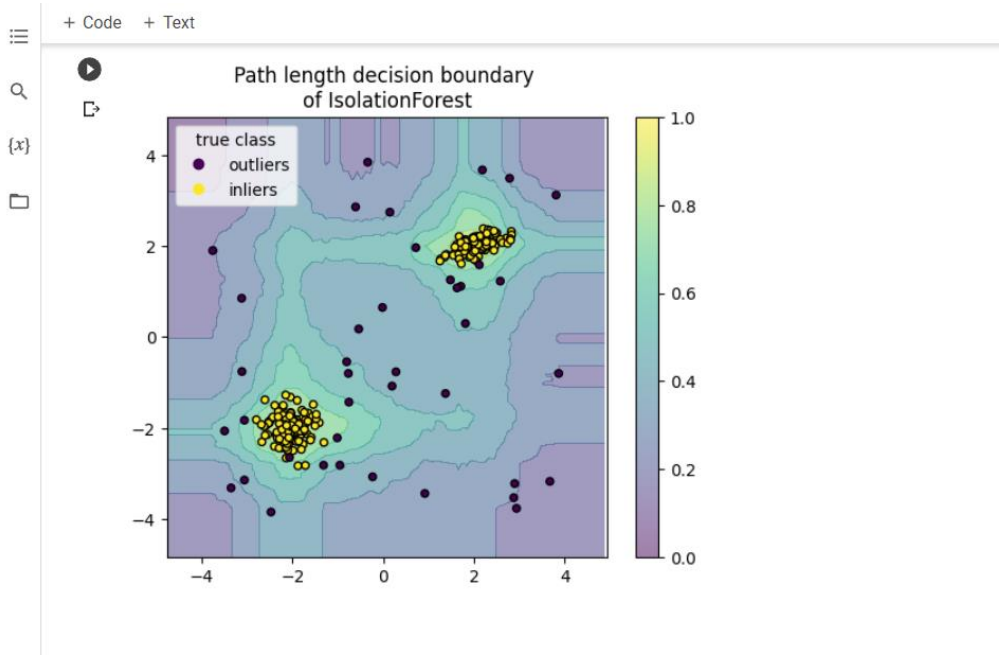


+ Code + Text

```
disp = DecisionBoundaryDisplay.from_estimator(
    clf,
    X,
    response_method="decision_function",
    alpha=0.5,
)
disp.ax_.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor="k")
disp.ax_.set_title("Path length decision boundary \nof IsolationForest")
plt.axis("square")
plt.legend(handles=handles, labels=["outliers", "inliers"], title="true class")
plt.colorbar(disp.ax_.collections[1])
plt.show()
```

Results and analysis

Result:



Analysis of the code:

1. The first block of code imports necessary libraries and generates a toy dataset. The dataset consists of two clusters of Gaussian-distributed points and some uniformly distributed outliers. The **train_test_split function** from scikit-learn is then used to split the data into training and test sets.
2. An Isolation Forest model is created and fitted to the training data using the fit method. The model's hyperparameters are set to max_samples=100 and random_state=0. The max_samples parameter controls the number of samples used to build each tree in the forest, and **random_state sets** the random seed for reproducibility.
3. Two plots are created to visualize the decision boundary of the Isolation Forest model. The **DecisionBoundaryDisplay class** from scikit-learn is used to generate the plots. The first plot shows the binary decision boundary obtained from the predict method of the model. The second plot shows the decision boundary obtained from the **decision_function** method, which returns the average path length of the samples in the trees of the forest. Both plots indicate the true class of each point with color-coded markers, and the legend distinguishes between inliers and outliers.
4. The plots are displayed using the show method of **the pyplot module** from Matplotlib.

In summary, this code demonstrates how to use the Isolation Forest algorithm in Python for anomaly detection. It generates a toy dataset, trains an Isolation Forest model on the data, and visualizes the decision boundary of the model in two different ways. The code is well-commented and organized, making it easy to understand and modify for different datasets and use cases.

Conclusion:

Anomaly detection is a critical task in various domains, such as fraud detection, cybersecurity, and medical diagnosis. It involves identifying rare and unusual data points that deviate significantly from the normal behavior of the system.

In this Python script, we demonstrated the use of the Isolation Forest algorithm for anomaly detection. Isolation Forest is a fast and efficient algorithm that constructs random decision trees and isolates anomalous points with shorter average path lengths. It works well for high-dimensional datasets and can handle both global and local anomalies.

We generated a toy dataset with two clusters of Gaussian distributed points and some uniformly distributed outliers, split it into training and test sets, and fit an Isolation Forest model to the training set. We then visualized the decision boundary of the model using the **DecisionBoundaryDisplay class**, which showed that the model could effectively separate inliers from outliers.

Overall, the Isolation Forest algorithm is a useful tool for anomaly detection tasks, and this script demonstrates its implementation in Python using the scikit-learn library. However, it is essential to note that the performance of the algorithm can depend on the dataset and its specific characteristics, and it is always advisable to evaluate the model's performance thoroughly on a validation set before deploying it to production.