The Pythoneers

Hidden Gems in Outlier Detection: 3 Powerful, Lesser-Known Methods

Uncover new tools for your data analysis toolkit with hands-on Python examples and explanations.

Thomas Konstantinovsky · Following

Published in The Pythoneers · 6 min read · Oct 14, 2024

Introduction

Outlier detection is a crucial part of data analysis, helping identify anomalies that can signal data errors, fraud, or rare events. While traditional methods like Z-score and IQR are widely used, there are lesser-known algorithms that provide **powerful** alternatives. In this post, I'll introduce three such methods: **Isolation Forest (iForest)**, **Local Outlier Probability (LoOP)**, and **Minimum Covariance Determinant (MCD)**. Each explanation will cover the intuition behind the method, how the algorithm works in detail, and a Python example using mock data.

1. Isolation Forest (iForest)

Isolation Forest is based on the concept of isolating points in a dataset. Unlike distance or density-based approaches, iForest works by randomly selecting a feature and splitting the dataset at a random value within the range of that feature. The main idea is that anomalies are “few and different,” making them easier to isolate than normal points, which are “many and alike.”

- **Algorithm Steps**:

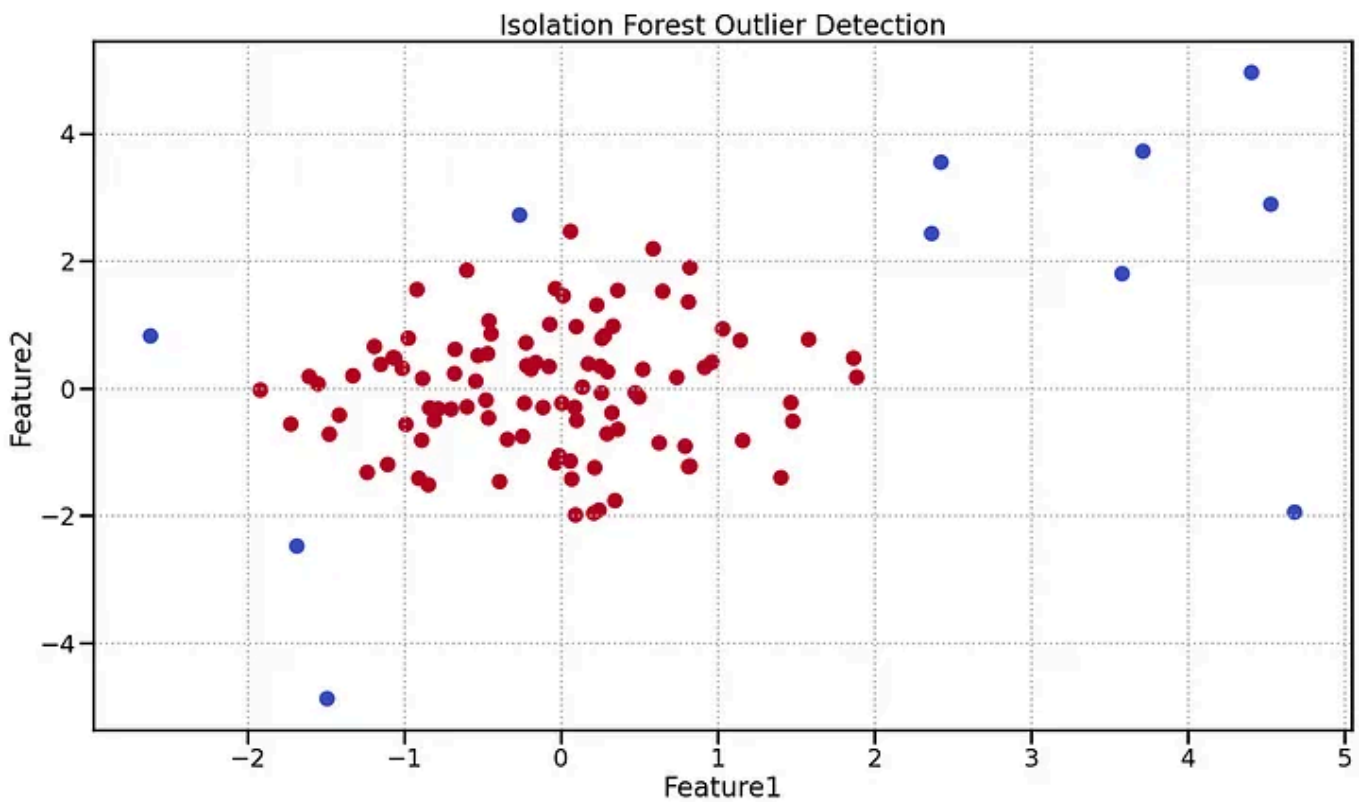
1. A random feature is selected.
2. A random split value is chosen between the minimum and maximum values of this feature.
3. The dataset is split into two partitions based on the split value.
4. Steps 1–3 are repeated until each data point is isolated (i.e., it’s alone in its partition).

The isolation process for normal points takes more splits, as they are clustered together. For anomalies, fewer splits are needed since they are separated from the majority of the dataset.

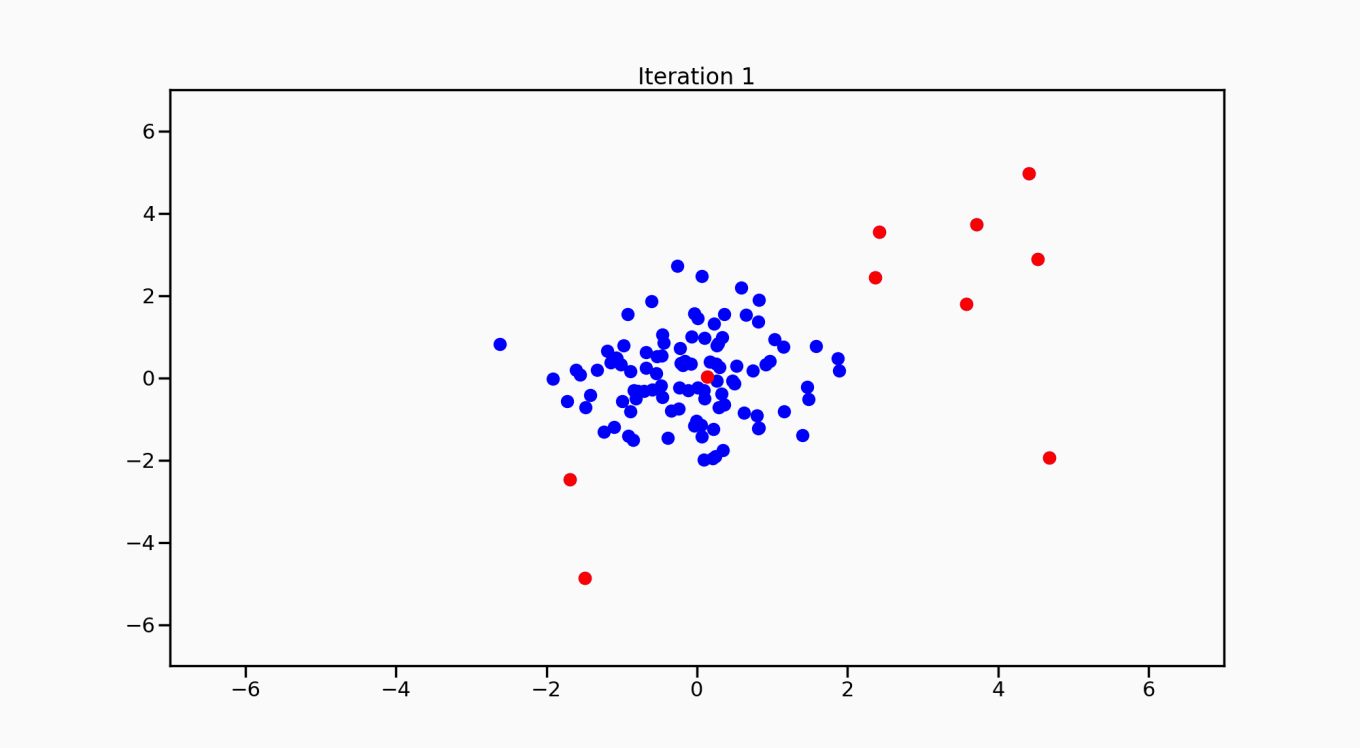
- **Anomaly Score Calculation**: The average path length of an observation (i.e., the number of splits required to isolate it) is used to calculate the anomaly score. Shorter paths indicate higher likelihoods of being anomalies.

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

# generating mock data
```

```
np.random.seed(42)
normal_data = np.random.normal(0, 1, (100, 2))  # use normal distributed data
outliers = np.random.uniform(low=-6, high=6, size=(10, 2))  # insert random outl
data = np.vstack((normal_data, outliers))
df = pd.DataFrame(data, columns=['Feature1', 'Feature2'])

# applying sklearn implementation of Isolation Forest
model = IsolationForest(contamination=0.1, random_state=42)
df['Anomaly'] = model.fit_predict(df[['Feature1', 'Feature2']])

# visualize the results
plt.scatter(df['Feature1'], df['Feature2'], c=df['Anomaly'], cmap='coolwarm')
plt.title("Isolation Forest Outlier Detection")
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.grid(lw=2,ls=':')
plt.show()
```



Results from Isolation Forest



Animation representing the itertaive progression of the IForest algorithm

In this example, we create a dataset consisting of normal points centered around (0,0) and a few anomalies far from this cluster. The Isolation Forest assigns an anomaly score based on how easily each point can be isolated. The scatter plot colors the anomalies differently from the normal points.

2. Local Outlier Probability (LoOP)

LoOP is an enhancement of the Local Outlier Factor (LOF), which itself measures the local density deviation of a point with respect to its neighbors. LoOP, however, refines this approach by calculating a probabilistic outlier score based on local densities, making it more adaptable and easier to interpret.

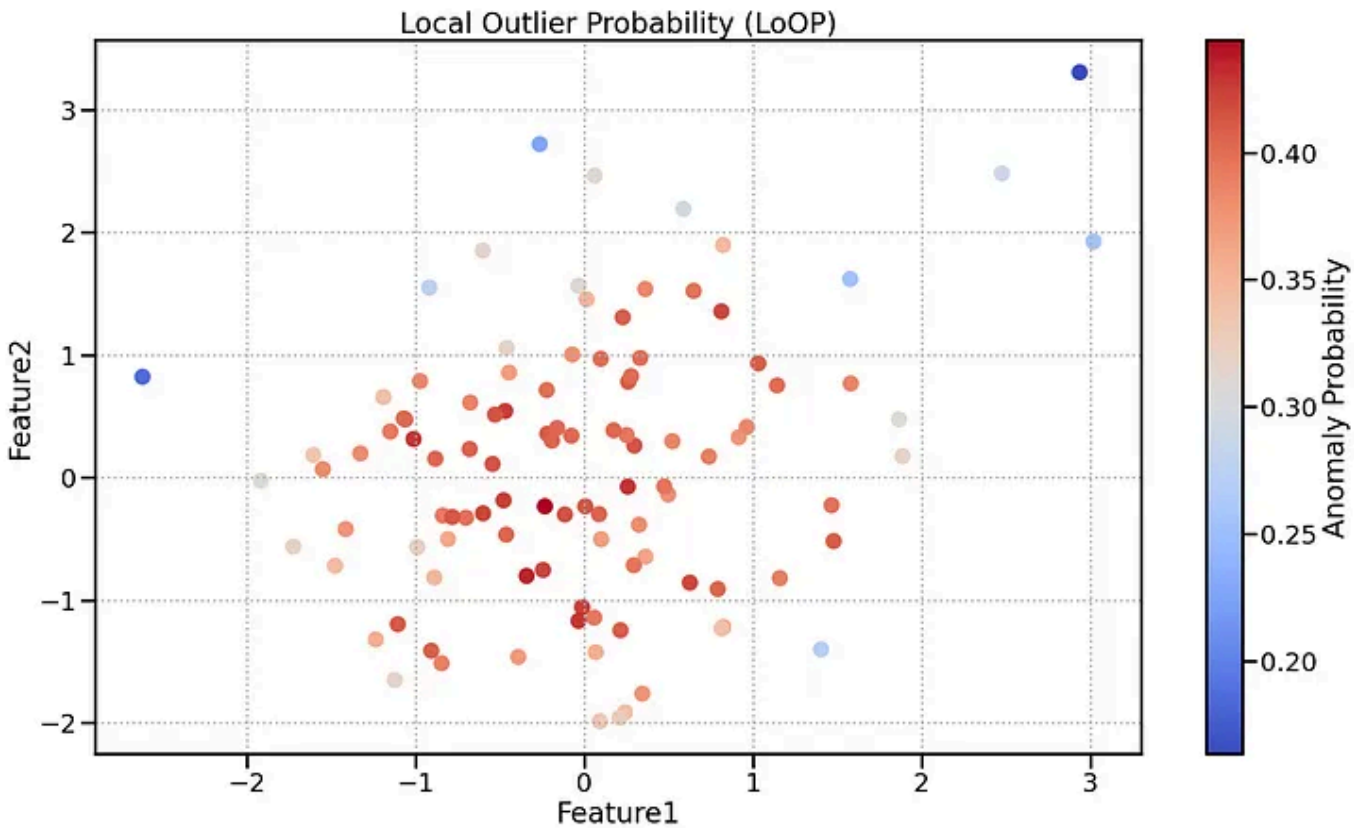
- **Algorithm Steps**:

1. For each point, calculate the local density using its k nearest neighbors.

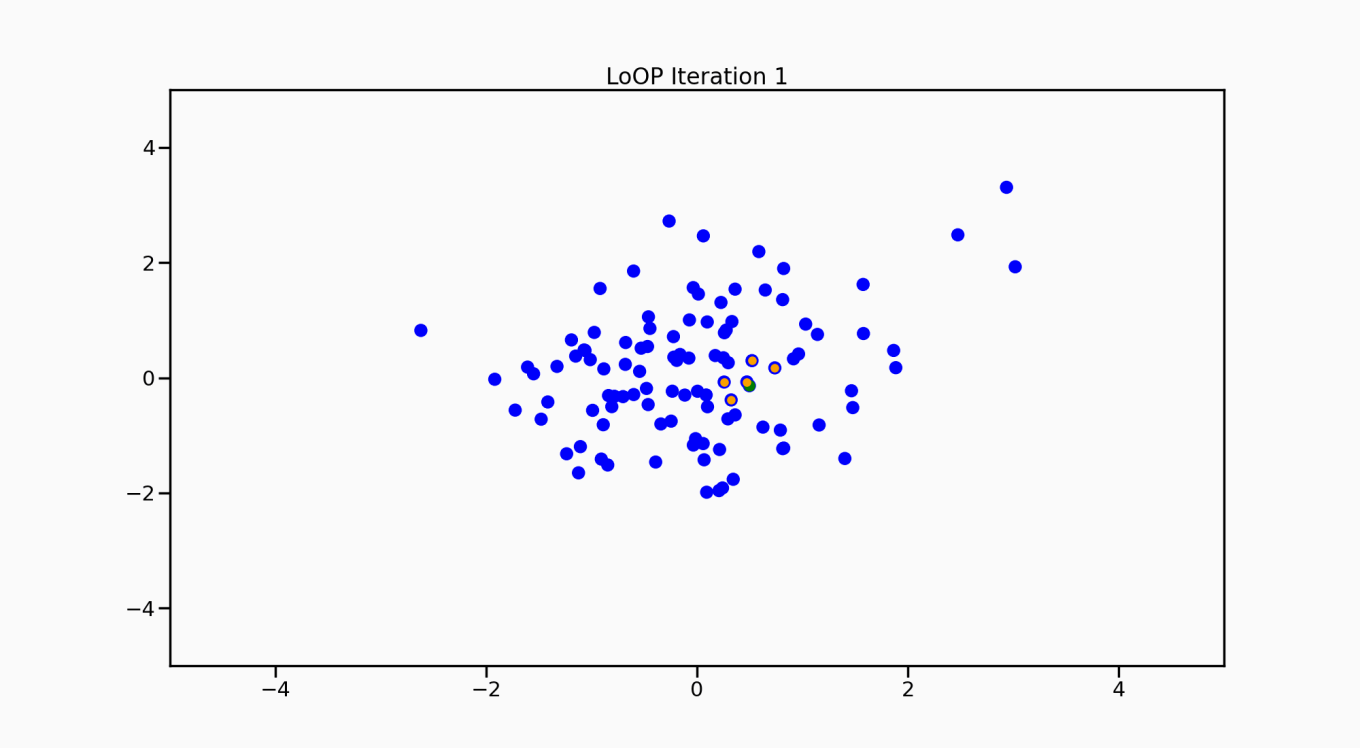
2. Compute the Local Reachability Distance (LRD), which is the inverse of the local density.
3. Determine the Probabilistic Local Outlier Factor (PLOF), which measures how the LRD of a point deviates from the LRDs of its neighbors.
4. Transform the PLOF into a probability score using a Gaussian distribution, ensuring the score is bounded between 0 (not an outlier) and 1 (strong outlier).

```
import numpy as np
import matplotlib.pyplot as plt

def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

def k_nearest_neighbors(X, point, k):
    distances = np.array([euclidean_distance(x, point) for x in X])
    return np.argsort(distances)[1:k + 1], distances[1:k + 1]  # Ignore self

def local_density(X, point_idx, k):
    neighbors, distances = k_nearest_neighbors(X, X[point_idx], k)
    return np.mean(distances)

def loop(X, k):
    densities = np.array([local_density(X, i, k) for i in range(len(X))])
    p_lof = np.zeros(len(X))

    for i in range(len(X)):
        neighbors, _ = k_nearest_neighbors(X, X[i], k)
        p_lof[i] = np.mean([densities[neighbor] for neighbor in neighbors]) / de

    prob_scores = 1 - np.exp(-p_lof ** 2 / 2)
    return prob_scores

np.random.seed(42)
normal_data = np.random.normal(0, 1, (100, 2))
outliers = np.random.uniform(low=-4, high=4, size=(5, 2))
data = np.vstack((normal_data, outliers))

k = 5
prob_scores = loop(data, k)
```

```
plt.scatter(data[:, 0], data[:, 1], c=prob_scores, cmap='coolwarm')
plt.colorbar(label='Anomaly Probability')
plt.title("Local Outlier Probability (LoOP)")
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.grid(lw=2,ls=':')
plt.show()
```



LoOP Result Example



Each time we take the 5 NN to each point to derive density, iteartion over all point

LoOP is advantageous because it accounts for the variability in density around each point, providing a more robust measure in cases where data density varies across the dataset.

3. Minimum Covariance Determinant (MCD)

MCD is a robust estimator for multivariate datasets that aims to find the subset of points with the smallest determinant of their covariance matrix. This subset represents the core structure of the dataset, allowing MCD to create a robust measure of central tendency and dispersion that isn't skewed by outliers.

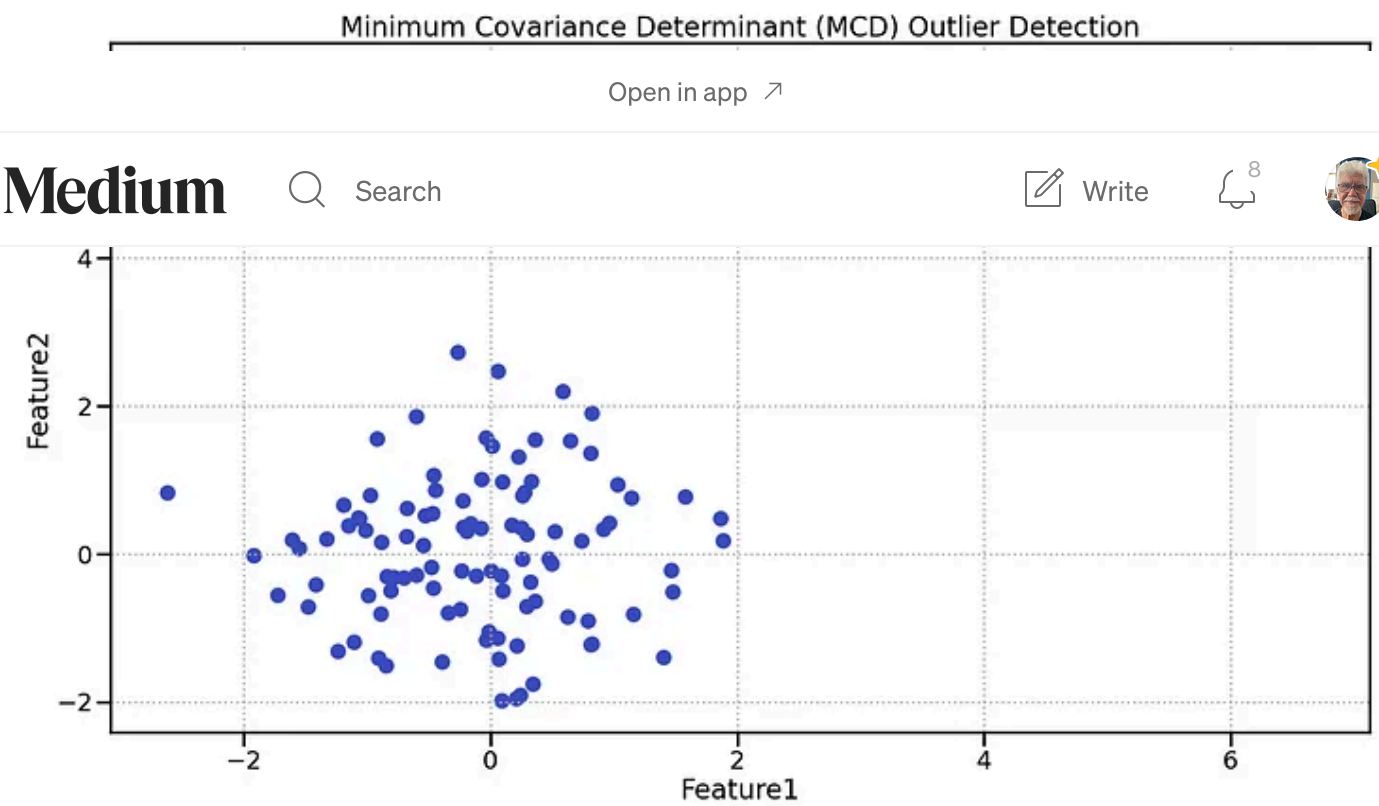
- **Algorithm Steps**:

1. Randomly select a subset of data points.
2. Calculate the covariance matrix and its determinant for this subset.

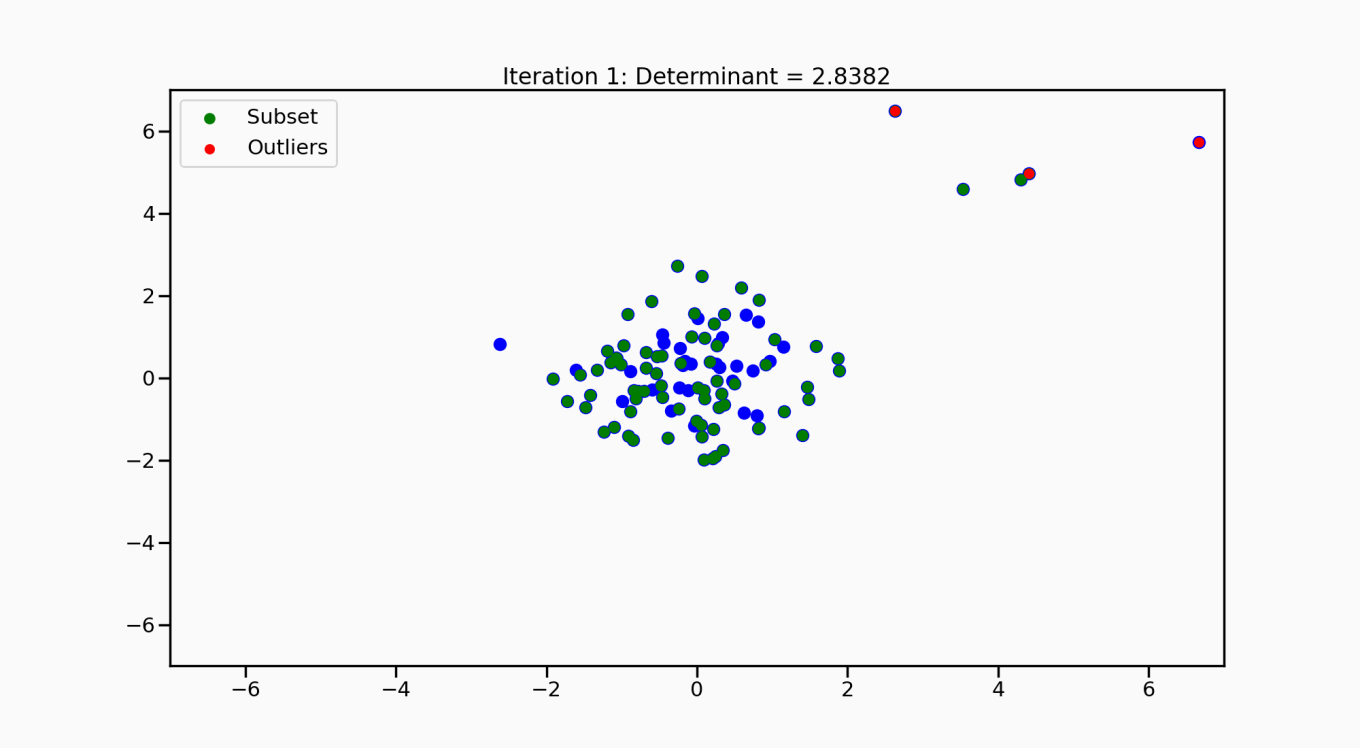
3. Repeat the process with different subsets, selecting the one with the smallest determinant.
4. Calculate the Mahalanobis distance for each observation based on this robust estimation. Points with distances significantly larger than the average are considered outliers.

```
import numpy as np
import matplotlib.pyplot as plt

def covariance_matrix(X):
    mean = np.mean(X, axis=0)
    return np.cov((X - mean).T)

def mahalanobis_distance(X, mean, cov_matrix):
    inv_cov = np.linalg.inv(cov_matrix)
    diffs = X - mean
    return np.sqrt(np.diag(np.dot(np.dot(diffs, inv_cov), diffs.T)))

# MCD implementation
def mcd(X, h):
    # here we randomly select subsets
    np.random.seed(42)
    best_det = float('inf')
    best_subset = None

    # testing 100 random subsets of h points
    for _ in range(100):
        subset = X[np.random.choice(X.shape[0], h, replace=False)]
        cov_matrix = covariance_matrix(subset)
        det = np.linalg.det(cov_matrix)
        if det < best_det:
            best_det = det
            best_subset = subset

    # finding the best subset's mean and covariance matrix
    best_mean = np.mean(best_subset, axis=0)
    best_cov = covariance_matrix(best_subset)

    # deriving the Mahalanobis distances for all points
    distances = mahalanobis_distance(X, best_mean, best_cov)
    return distances
```

```
np.random.seed(42)
normal_data = np.random.normal(0, 1, (100, 2))  # Normally distributed data
outliers = np.random.multivariate_normal([5, 5], [[1, 0.5], [0.5, 1]], 5)  # Out
data = np.vstack((normal_data, outliers))

# apply MCD to detect outliers
h = 75  # we use half the data points to estimate the covariance
distances = mcd(data, h)

# threshold to determine outliers (2 standard deviations away)
threshold = np.percentile(distances, 97.5)
outliers_pred = distances > threshold

plt.scatter(data[:, 0], data[:, 1], c=outliers_pred, cmap='coolwarm')
plt.title("Minimum Covariance Determinant (MCD) Outlier Detection")
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.grid(lw=2,ls=':')
plt.show()
```



MCD Result Example



Detriminat Claculation And Comparison Agianst Remaining Points

The MCD approach works well in multivariate cases where traditional mean and covariance estimates are skewed by outliers. It isolates the core distribution, ensuring that anomalies are accurately detected.

Conclusion

These lesser-known outlier detection methods offer diverse and powerful tools for uncovering anomalies in complex datasets. Whether you need a probabilistic approach like LoOP or a multivariate method like MCD, these techniques provide robustness beyond common methods. By experimenting with these algorithms, you can uncover insights that may go unnoticed with traditional approaches.

Machine Learning | Data Science | Data Analysis | Outlier Detection | Mathematics

Published in The Pythoneers

13.2K Followers · Last published Mar 26, 2025

Following

Your home for innovative tech stories about Python and its limitless possibilities. Discover, learn, and get inspired.

Written by Thomas Konstantinovsky

321 Followers · 10 Following

Following

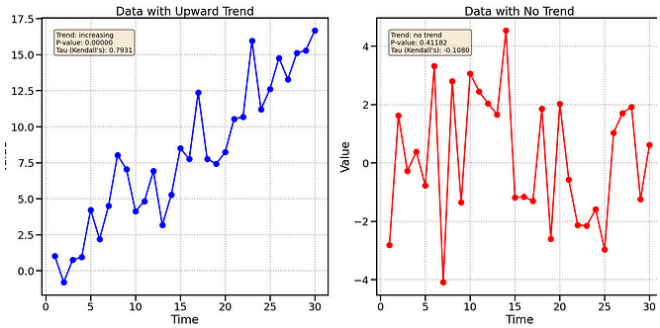
PhD Student & Data Scientist driven by curiosity—on a relentless quest to uncover the hidden patterns that shape our world.

No responses yet

Alex Mylnikov

What are your thoughts?

More from Thomas Konstantinovsky and The Pythoneers



In The Pythoneers by Thomas Konstantinovsky

5 Underrated Statistical Tests You Didn't Know You Needed

From TCR Repertoires to Stock Prices, These Hidden Gems Can Transform Your Analysis

Dec 30, 2024 575 6

In The Pythoneers by Abhay Parashar

How to Explain Each Core Machine Learning Model in an Interview

From Regression to Clustering to CNNs: A Brief Guide to 25+ Machine Learning Models

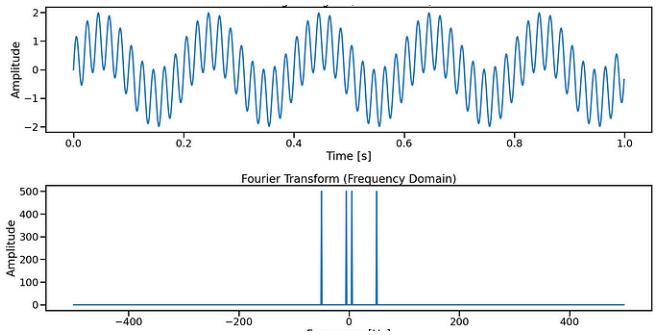
Mar 12 1.8K 21

In The Pythoneers by Aashish Kumar

9 Modern Python Libraries You Must Know in 2025! 🚀

I have discovered these few game changer modern set of libraries that you should must...

Mar 2 309 3



In The Pythoneers by Thomas Konstantinovsky

Wavelet Transform: A Practical Approach to Time-Frequency...

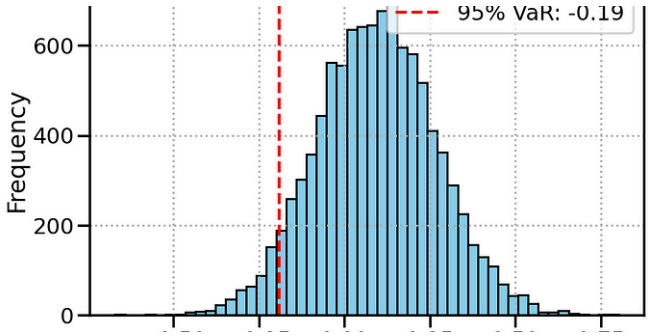
Unlocking the Power of Wavelet Transforms: A Guide to Time-Frequency Signal Analysis...

Oct 5, 2024 235 4

See all from Thomas Konstantinovsky

See all from The Pythoneers

Recommended from Medium



In The Pythoneers by Thomas Konstantinovsky

Monte Carlo Simulation: Ideas and Examples for Advanced...

Some less-known examples of how to leverage Monte Carlo simulation for real-...

Nov 24, 2024 220 2

Alice Yang

Mastering Conditional Formatting in Excel Using Python: A...

Conditional formatting is a valuable feature in spreadsheet applications like Microsoft Exc...

Mar 6 62

In Data Science Collective by Ryan Burn

Constructing Default Priors for Bayesian Hypothesis Testing: A...

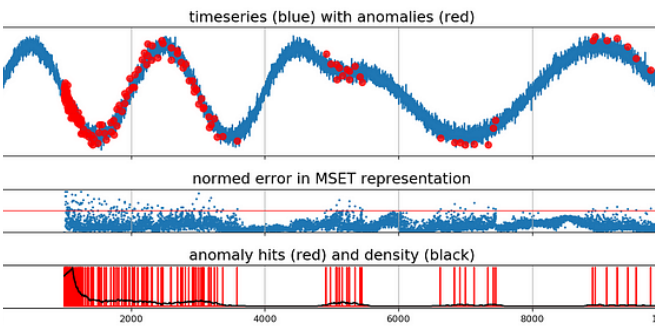
Consider these birth records from Paris collected between 1745 and 1770:

In Psyc Digest by Alessia Fransisca

The 1-Minute Introduction That Makes People Remember You...

A Behavioral Scientist's Trick to Hack the "Halo Effect"

3d ago 6



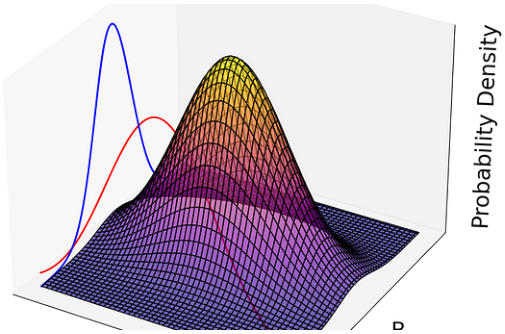
In Writing in the World of Artificial Int... by Abish ...

Anomaly Detection on Time Series with MSET-SPRT in Python

In the world of anomaly detection, especially for complex systems like industrial machiner...

Mar 25 26 1

Mar 19 5.2K 123



Irene Markelic, PhD

Understanding the Bivariate Normal Distribution

A Mathematical Derivation of its Probability Density Function

Mar 26 109 3

See more recommendations