Overview

This report serves as a brief overview of the three methods for anomaly detection implemented in my TROPOMI web service. We will describe the algorithms backing each method, explain their hyperparameters, and link the relevant software documentation for each method.

Note: Instructions for setting up and running the code are provided in the README file in my Git repository. This report just explains the possible methods you can use and their hyperparamters.

1 The Local Outlier Factor

Hyperparameters: For this method, you must specify the following in the `config.yml` configuration file:

(i) `spreadStatistic`: Either the *Interquartile Range* (use `'IQR'`), the *Standard Deviation* (use `'StandardDeviation'`), or the *Mean Absolute Deviation* (use `MAD`).

(ii) `threshold`: Some positive real number. Suggested values are in $[1.0, 3.0]$.

(iii) `numNeighbors`: Some positive integer. Suggested values are in $[1, 40]$. The number of points surrounding a value that will be used by the algorithm to calculate density.

(iv) `algorithm`: The algorithm used to find the nearest neighbors to a point. Either the `ball tree` algorithm (use `'ball_tree'`) or the `KD Tree` algorithm (use `'kd_tree'`).

(v) `leafSize`: Some positive integer. Suggested values are in $[10, 40]$. This is a hyperparameter for the selected `algorithm`.

(vi) `metric`: The norm used to calculate distance between points. Recommended choices are ℓ_1-Norm (use `manhattan`), ℓ_2-Norm (use `euclidean`), or ℓ_p-Norm for $p > 2$ (use `'minkowski'`).

(vii) `p`: Some positive integer. Suggested values are in $[1, 4]$. This is the norm to use for the `metric` specified above. Use $p = 1$ for `'metric': 'manhattan'`, use $p = 2$ for `'metric': 'manhattan'`, and use $p > 2$ for `'metric': 'minkowski'`.

The Method: This method works by taking the response and removing all of the values that are within a value of `threshold` $\times$ `standardDeviation` of the mean. The remaining response values are plotted on a spatial map and the points of unusually high or low density are classified as anomalies. A point's density is calculated based on the reciprocal of the sum of squared distances between that point and its `numNeighbors`

nearest spatial neighbors, where distance is defined by the `metric` hyperparameter.

Visualization: An image depicting the local outlier factor calculating the density of point A from its 3 nearest-neighbors (and using the ℓ_2-Norm for distance is shown in Figure 1 below.

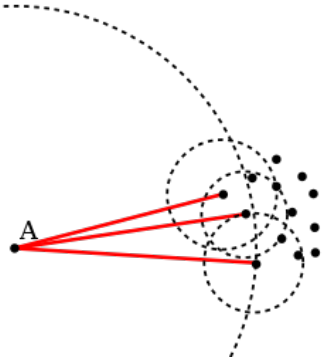

Figure 1: The Local Outlier Factor Finds Points with Unusual Density.

This method does a great job of finding anomalies, but tends to classify some normal points as anomalies. It tends toward Type I error.

Relevant Documentation: For Python, see the function in the `scikit-learn` package. For `R`, see the function in the `dbscan` package.

2 The Isolation Forest

Hyperparameters: For this method, you must specify the following in the `config.yml` configuration file:

(i) `spreadStatistic`: Either the *Interquartile Range* (use `'IQR'`), the *Standard Deviation* (use `'StandardDeviation'`), or the *Mean Absolute Deviation* (use `MAD`).

(ii) `threshold`: Some positive real number. Suggested values are in $[1.0, 3.0]$.

(iii) `numEstimators`: Some positive integer. Suggested values are in $[50, 200]$. The number of times to perform the isolation cuts starting from a random position.

(iv) `bootstrap`: A boolean specifying whether sampling should occur with or without replacement from the response during each random isolation cut. If `True`, sampling with replacement occurs. Otherwise, sampling without replacement occurs.

The Method: This method works by taking the response and removing all of the values that are within a value of `threshold` $\times$ `standardDeviation` of the mean. The remaining response values are plotted on a spatial map. Then, for each remaining point (call the chosen point P_i, say) the following process is executed `numEstimator` times:

1. A sample is taken (replacement or no replacement depends on the `bootstrap` hyperparameter) from the remaining response values. The sample size is the number of the remaining response values. This sample is plotted spatially.
2. A random place on the spatial grid is chosen for a vertical or horizontal "cut" that divides the plot in half.
3. The remaining half of the plot containing P_i is determined.
4. If P_i is the only point in the remaining half of the plot, the number of cuts to get to P_i is returned.
5. Steps 2, 3, and 4 continue recursively until P_i is isolated.

The output is `numEstimator` numbers of cuts taken to isolate each point P_i that is in the remaining response values. Points with the fewest total number of cuts to isolate are considered the most anomalous.

Visualization: An image depicting point x_1 being classified as "usual" and point x_0 being classified as "anomalous" using the isolation forest is shown in Figure 2 below.

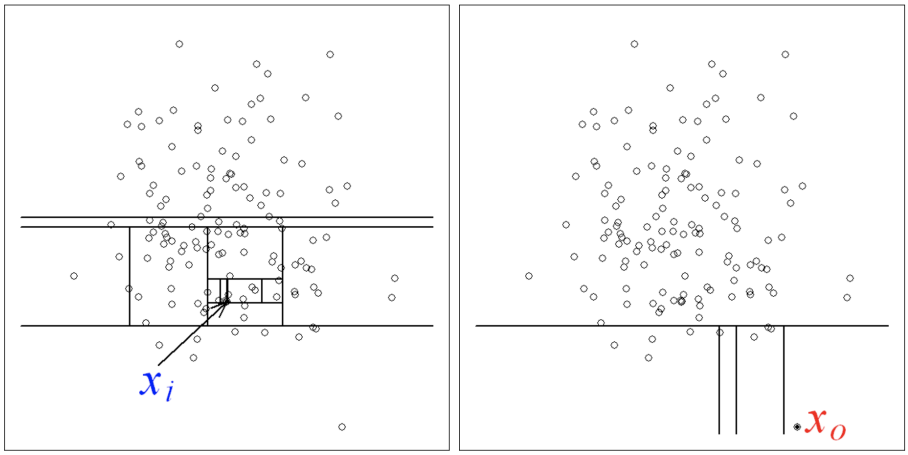

Figure 2: The Isolation Forest Partitions Spatial Regions to Find Outliers.

This method behaves very similarly to the local outlier factor. It tends toward Type I error.

Relevant Documentation: For Python, see the function in the `scikit-learn` package. For `R`, see the function in the `isotree` package.

3 The Autoencoder Neural Network

Hyperparameters: For this method, you must specify the following in the `config.yml` configuration file:

(i) `depth`: Some positive integer. Suggested values are in $[5, 50]$. This is the number of "hidden" encoding layers to include.

(ii) `anomalyScoreCutoff`: Some positive real number. Suggested values are in $[3.0, 5.0]$, but this cutoff really depends on the distribution of anomaly scores from the autoencoder.

The Method: This method works by trying to predict the entire set of response values using only a small, encoded subset of the features in the data. The response values that are most unlike the others tend to be excluded from the encoded features, and they are classified as anomalies. So, the points that do not survive the encoding reduction receive higher anomaly scores.

The anomaly scores for each response value are visualized in a histogram (in my code, this histogram gets saved to `MATH582/software/analyze/static/images/anomalyScoresPlot.png`). The use must look at this plot and choose some cutoff score S^* such that – for all anomaly scores greater than S^* – the corresponding observations are classified as anomalies. All other observations are considered normal.

Visualization: An image depicting the autoencoder reducing the full response (far left) to a small encoded subset (middle) and predicting the full response from the encoded subset (right) is shown in Figure 3 below.

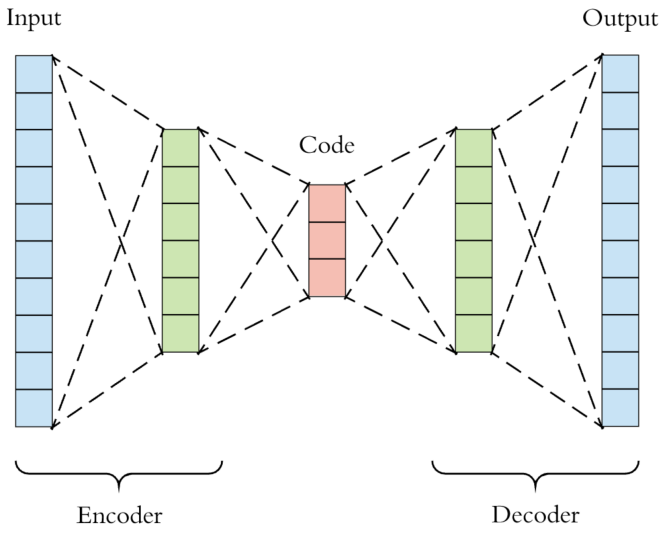

Figure 3: The Autoencoder Estimates Input Features from a Small Encoded Subset.

This method is also effective in determining outliers, but can fail to find some obvious outliers without choosing a responsible cutoff score. In general, it tends toward Type II error.

Relevant Documentation: For Python, see the `AutoEncoder` class in the `PyOD` package. For `R`, see the `autoencoder` function in the `R` interface to the `Keras` package.