Outlier Detection in Modeva

July 5, 2025

Outline

What are Outliers?

- **Definition:** Data points that significantly deviate from the rest of the data
- **Causes:**
  - Measurement errors
  - Data corruption
  - Natural variation (rare but valid observations)
  - Different data generation processes
- **Impact:**
  - Skew statistical analyses
  - Bias model training
  - Lead to incorrect conclusions
  - Sometimes represent important edge cases

Importance of Outlier Detection

- **Data preprocessing:** Crucial step for quality data
- **Model robustness:** Improves stability and generalizability
- **Feature engineering:** Informs transformation decisions
- **Domain insights:** May reveal important anomalies
    - Fraud detection in financial transactions
    - Network intrusion detection
    - Medical diagnosis of rare conditions
    - Manufacturing quality control

Outlier Detection in MoDeVa

The `DataSet` class in MoDeVa implements three outlier detection strategies:

- `DataSet.detect_outlier_cblof`
  - Cluster-Based Local Outlier Factor (CBLOF)
  - Uses clustering to identify anomalies
- `DataSet.detect_outlier_isolation_forest`
  - Based on the Isolation Forest algorithm
  - Isolates observations through random partitioning
- `DataSet.detect_outlier_pca`
  - PCA-based outlier detection
  - Uses Mahalanobis distance or error reconstruction

CBLOF Overview

`DataSet.detect_outlier_cblof` implements the CBLOF algorithm:

- **Originally proposed by:** He et al. (2003)
- **Core idea:** Use clustering to identify outliers
- **Assumption:** Outliers are far from cluster centers or belong to small clusters
- **Advantages:**
    - Intuitive approach
    - Considers local data structure
    - Can handle different cluster densities

CBLOF Methodology

Step 1: Clustering

- Partition data into clusters using K-means or Gaussian Mixture Model
- Classify clusters into two categories:
  - Large clusters: Contain many data points
  - Small clusters: Contain few data points
- Classification based on cluster size threshold

Step 2: CBLOF Score Calculation

- **For points in large clusters:**
  - Compute Euclidean distance to own cluster centroid
- **For points in small clusters:**
  - Compute Euclidean distance to nearest large cluster centroid

CBLOF Score Weighting

- **Optional weighting:** Multiply score by cluster size
  - Default: No multiplication (raw distance)
  - With multiplication: Emphasizes outliers in larger clusters
- **Final score:** Higher values indicate greater likelihood of being an outlier
- **Comprehensive measure:** Considers both
  - Distance within a cluster
  - Relative distances to neighboring clusters

Implementing CBLOF in Modeva

```
from modeva import DataSet
dataset = DataSet(data)
# Detect outliers using CBLOF
outlier_scores = dataset.detect_outlier_cblof(
    n_clusters=5,  # Number of clusters
    cluster_method='kmeans',  # Clustering algorithm
    alpha=0.9,  # Large/small cluster threshold
    beta=5,  # Minimum size ratio between large and small clusters
    use_weights=False  # Whether to weight by cluster size
)
# Higher scores indicate greater likelihood of being an outlier
print(f"Outlier scores: {outlier_scores}")
```

- CBLOF is effective when your data naturally forms clusters
- Adjust n_clusters based on your domain knowledge
- The alpha parameter controls the threshold for large vs. small clusters

Isolation Forest Overview

`DataSet.detect_outlier_isolation_forest` implements the Isolation Forest algorithm:

- **Core idea:** Isolate observations through random partitioning
- **Assumption:** Outliers are few and different, thus easier to isolate
- **Advantages:**
  - Efficient for high-dimensional data
  - Low computational complexity: $O(n \, log \, n)$
  - Does not rely on distance or density measures
  - Handles various data distributions

Isolation Forest Methodology

Step 1: Building Isolation Trees

- Randomly select a feature
- Randomly select a split value between feature's min and max
- Recursively partition data
- Continue until:
  - Node contains only one instance, or
  - All data at node have same values

Step 2: Anomaly Score Calculation

- Compute average path length to isolate each observation
- Shorter path length $\rightarrow$ easier to isolate $\rightarrow$ likely outlier
- Longer path length $\rightarrow$ harder to isolate $\rightarrow$ likely normal

Implementing Isolation Forest in MoDeVa

Basic Usage

```
# Detect outliers using Isolation Forest
outlier_scores = dataset.detect_outlier_isolation_forest(
    n_estimators=100,  # Number of isolation trees
    max_samples='auto',  # Number of samples to draw for each tree
    contamination=0.1,  # Expected proportion of outliers
    random_state=42  # For reproducibility
)
# Higher scores indicate greater likelihood of being an outlier
print(f"Outlier scores: {outlier_scores}")
```

- **Note:** MoDeVa's implementation is a wrapper of scikit-learn's `IsolationForest`
- The contamination parameter represents the expected proportion of outliers
- Increasing n_estimators improves stability but increases computation time

PCA-Based Methods Overview

`DataSet.detect_outlier_pca` implements PCA-based outlier detection:

- **Core idea:** Use dimensionality reduction to identify anomalies
- **Two approaches implemented:**
  1. Mahalanobis distance
  2. Error reconstruction
- **Advantages:**
  - Accounts for feature correlations
  - Reduces dimensionality of high-dimensional data
  - Can detect complex anomalies

Mahalanobis Distance Approach

- **Definition:** Statistical distance that accounts for correlations in data
- **Process:**
  1. Apply PCA to transform data
  2. Calculate Mahalanobis distance in PCA space:

$$D_M^2 = \sum_{i=1}^{k} \frac{z_i^2}{\lambda_i}$$

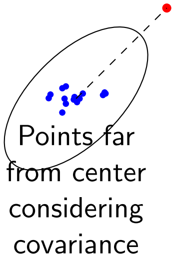
  where:
  - z_i = i-th principal component score
  - λ_i = i-th eigenvalue (variance along PC)
  - k = number of principal components used
- **Interpretation:** Higher distance indicates potential outlier

Error Reconstruction Approach

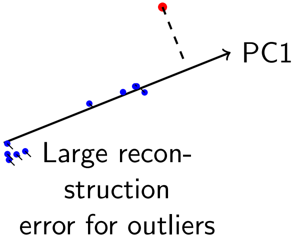
- **Definition:** Measure dissimilarity between original data and its PCA reconstruction
- **Process in Modeva:**
  1. Apply PCA to transform data
  2. Fit XGBoost model between principal components and original features
  3. Use model to reconstruct original features: X_{new}
  4. Calculate reconstruction error: $X - X_{new}$
  5. Compute Mahalanobis distance of reconstruction error as final score
- **Special case:** If reconstruction errors are independent, score reduces to mean squared reconstruction error

PCA-Based Methods Intuition

Mahalanobis Distance



Error Reconstruction



- Both methods leverage principal component analysis to identify outliers
- Mahalanobis accounts for correlation structure in the data
- Reconstruction identifies points that don't fit the principal patterns

Implementing PCA-Based Outlier Detection in MoDeVa

Basic Usage

```
# Detect outliers using PCA-based methods
outlier_scores = dataset.detect_outlier_pca(
    n_components=2,  # Number of principal components
    method='mahalanobis',  # 'mahalanobis' or 'reconstruction'
    reconstruction_model='xgboost'  # Model for reconstruction
)

# Higher scores indicate greater likelihood of being an outlier
print(f"Outlier scores: {outlier_scores}")
```

- Choose between 'mahalanobis' and 'reconstruction' methods
- Select n_components based on explained variance in your data
- For reconstruction, 'xgboost' is used by default to model the relationship

Comparing Outlier Detection Methods

Aspect	CBLOF	Isolation Forest	PCA-Based
Core approach	Clustering	Random partitioning	Dimensionality reduction
Computational complexity	$O(n^2)$ for K-means	$O(n \log n)$	$O(n^2)$ for PCA
Handles high dimensions	Moderate	Very good	Good
Considers local structure	Yes	No	Partially
Interpretability	High	Medium	Medium to high
Sensitivity to parameters	High	Medium	Medium
Handles mixed data	No	Yes	No

When to Use Each Method

Use CBLOF when:

- Data naturally forms clusters
- Local data structure is important
- Interpretability is a priority
- Dataset is small to medium-sized

Use Isolation Forest when:

- Dealing with high-dimensional data
- Computational efficiency is important
- Data distribution is complex or unknown
- Dataset contains mixed data types

Use PCA-based methods when:

- Feature correlations are important
- Data has a linear structure
- Statistical interpretation is desired
- Dimension reduction is beneficial

Best Practices for Outlier Detection

1. **Understand your data:** Examine distributions before choosing a method
2. **Try multiple methods:** Different algorithms may detect different types of outliers
3. **Parameter tuning:** Adjust parameters based on domain knowledge
4. **Visualization:** Plot outlier scores and examine detected outliers
5. **Domain validation:** Verify if detected outliers make sense in context
6. **Conservative approach:** Set thresholds carefully to avoid false positives
7. **Ensemble methods:** Combine multiple approaches for robust detection
8. **Feature selection:** Consider applying outlier detection to subsets of features

Complete Outlier Detection Workflow

```
cblof_scores = dataset.detect_outlier_cblof(
    n_clusters=5, cluster_method='kmeans')
isolation_scores = dataset.detect_outlier_isolation_forest(
    n_estimators=100, contamination=0.1)
pca_scores = dataset.detect_outlier_pca(
    n_components=5, method='mahalanobis')
# Normalize scores for comparison
def normalize(scores):
    return (scores - np.min(scores)) / (np.max(scores) - np.min(scores))
norm_cblof = normalize(cblof_scores)
norm_isolation = normalize(isolation_scores)
norm_pca = normalize(pca_scores)
combined_scores = (norm_cblof + norm_isolation + norm_pca) / 3 # avg. score
threshold = 0.9 # Set threshold and identify outliers
outlier_indices = np.where(combined_scores > threshold)[0]
```

Handling Detected Outliers

After identifying outliers, consider these approaches:

- **Remove:** Delete outliers if they represent errors
- **Transform:** Apply transformations to reduce outlier impact
  - Log transformation
  - Winsorization (capping extreme values)
  - Robust scaling
- **Separate modeling:** Create special models for outlier cases
- **Use robust methods:** Employ algorithms less sensitive to outliers
  - Robust regression
  - Tree-based methods
  - Robust PCA
- **Keep and monitor:** Sometimes outliers contain valuable information

Summary

- **Outlier detection** is crucial for data quality and model performance
- **Modeva provides three approaches:**
  - CBLOF: Cluster-based approach for local outlier detection
  - Isolation Forest: Efficient random partitioning approach
  - PCA-based: Using Mahalanobis distance or error reconstruction
- **Method selection** depends on:
  - Data characteristics and structure
  - Computational constraints
  - Interpretability requirements
- **Best practice:** Combine multiple methods and validate with domain knowledge