Outlier Detection & Treatment for Machine Learning Models

Vaibhav Tummalapalli

Atlanta, USA.

Abstract:
Outliers can significantly impact the performance of machine learning models by skewing data distributions and introducing noise. This paper explores various techniques for outlier detection and treatment, with a focus on Gaussian and non-Gaussian distributions, domain-specific limits, and visualizations. Practical implementation tips and Python code examples are provided to facilitate adoption in real-world scenarios

Keywords: Outlier Detection, Machine Learning, Python, IQR, Standard Deviation, KS Test, Median Absolute Deviation, Propensity Models.

I. INTRODUCTION

Exploratory data analysis has long emphasized the importance of detecting outliers through visual and distribution-based methods [1]. These outliers are data points that deviate significantly from the general trend or distribution of most observations. These may result from measurement error, rare events, or sampling anomalies, as explored in depth by Barnett and Lewis [3]. While outliers might represent valuable insights in some cases (e.g., identifying fraud or detecting rare phenomena), they can also distort statistical analyses and machine learning models, leading to biased results and poor predictive performance.

In statistical analyses, outliers may affect measures like the mean, variance, and correlation, skewing interpretations and conclusions. Similarly, in machine learning, the presence of outliers can lead to unstable models that either overfit the data or fail to generalize effectively to unseen samples. For example, outliers can disproportionately influence algorithms sensitive to distance metrics, such as k-Nearest Neighbors (k-NN) and Support Vector Machines (SVMs), or cause decision trees to create unbalanced splits.
Addressing outliers is a critical preprocessing step in the data pipeline, particularly for tasks like predictive modeling. Effective outlier handling involves:

- **Detection**: Identifying potential outliers using statistical methods (e.g., z-scores, interquartile ranges), machine learning techniques (e.g., isolation forests, DBSCAN), or domain-specific rules based on business knowledge.
- **Treatment**: Deciding how to handle the identified outliers, which may involve removal, transformation, or replacement using capping, imputation, or scaling techniques.

This paper provides a comprehensive overview of techniques for detecting and addressing outliers in datasets. It categorizes these approaches into three main scenarios:

- **Gaussian Data**: Assuming the data follows a normal distribution, techniques like z-scores and standard deviation thresholds are applied to detect deviations.
- **Non-Gaussian Data**: For data that does not conform to a normal distribution, robust techniques such as interquartile ranges (IQR), median-based approaches are explored.
- **Domain-Specific Scenarios**: In cases where contextual knowledge about the data is available, domain-driven rules and thresholds are utilized to identify and treat outliers effectively.

For each scenario, the paper provides Python code implementations, ensuring practical applicability for readers. Additionally, it evaluates the impact of outlier treatment on downstream tasks, highlighting scenarios

were retaining or transforming outliers can enhance model performance. By bridging theory and implementation, this paper equips practitioners with the tools needed to address outliers effectively, ensuring stable and unbiased results in statistical and machine learning workflows. This comprehensive approach emphasizes the importance of thoughtful preprocessing, enabling better insights and decisions from data analysis pipelines

II. OUTLIER DETECTION & TREATMENT

Outlier detection is a critical preprocessing step in machine learning, as the presence of extreme values can distort model estimates, bias parameter learning, and reduce overall predictive performance. The choice of outlier detection and treatment methods should be informed by both the statistical distribution of the variables and the contextual domain knowledge. Rather than omitting anomalous records, which can lead to information loss, a commonly adopted practice involves replacing extreme values with the nearest acceptable boundary. This approach helps maintain dataset integrity while minimizing the influence of outliers on model training.

A. Gaussian Distribution

For features that approximately follow a normal distribution, outliers can be identified using z-score or modified z-score methods. These methods quantify how far a data point deviates from the mean (μ), scaled by the standard deviation (σ), under the assumption that the data adheres to a bell-shaped curve. In classical statistical process control, thresholds of ±3σ are often used to flag anomalies [2]. Observations lying outside this range are considered statistically rare and are candidates for investigation or treatment.

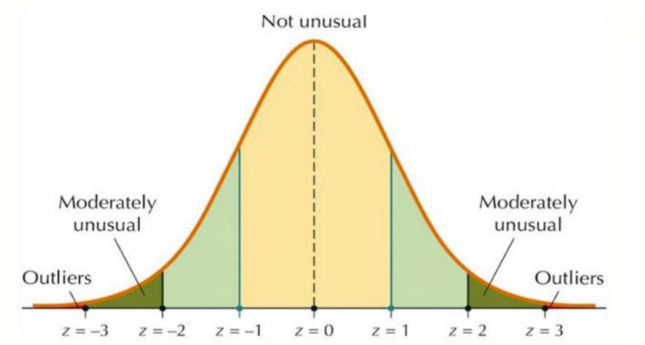

Fig 1. Gaussian Distribution Curve

Let x be a continuous variable. A standard z-score is computed as:

$$z = \frac{x - \mu}{\sigma}$$

Where:

- μ is the sample mean,
- σ is the standard deviation of the variable.

Values where |z| > 3 are often labeled as outliers. For datasets with heavy tails or where the assumption of normality is weak, alternative robust measures such as the interquartile range (IQR) or median absolute deviation (MAD) may be more appropriate.

Furthermore, domain-specific thresholds can supplement statistical criteria—especially in industrial, biomedical, or automotive settings where extreme values may still carry valid operational meanings.

Steps for Gaussian-Based Outlier Treatment:

- **Compute Central Tendency and Dispersion:** Calculate the mean μ and standard deviation σ for the variable under analysis.
- **Identify Outlier Thresholds:** Define the acceptable range for data points using the interval [μ−3σ, μ+3σ]. Any observation falling outside this range is considered an outlier under the empirical rule of normal distribution.
- **Apply Capping or Flooring:** Replace values below μ−3σ with the lower boundary, and values above μ+3σ with the upper boundary. This preserves the number of observations while reducing the influence of extreme values on the model.

Python Implementation [6]:

```
import numpy as np

# Example data
data = np.random.normal(40, 10, 1000)

# Calculate mean and standard deviation
mean = np.mean(data)
std = np.std(data)

# Define bounds
lower_bound = mean - 3 * std
upper_bound = mean + 3 * std

# Cap and floor outliers
data_clipped = np.clip(data, lower_bound, upper_bound)
```

B. Non-Gaussian

In many real-world scenarios, data deviates from the ideal bell-shaped curve and exhibits skewness, heavy tails, or multimodal characteristics. In such cases, traditional z-score-based methods—relying on the mean and standard deviation—can be misleading, as these metrics are sensitive to extreme values. Instead, robust statistical techniques such as the Interquartile Range (IQR) and Median Absolute Deviation (MAD) offer more reliable alternatives for outlier detection in non-Gaussian distributions.

Skewed Data Distributions

Figure 2 illustrates examples of skewed distributions. The left panel depicts left-skewed (negatively skewed) data, while the right panel shows a right-skewed (positively skewed) distribution. For comparison, the center panel displays a symmetric distribution resembling a Gaussian profile. In such skewed distributions, the mean and standard deviation are not ideal indicators of central tendency and spread, making percentile-based methods preferable.

Interquartile Range (IQR) Method

The IQR method uses the middle 50% of the data, which is less influenced by extreme values. It defines outliers relative to the interquartile spread.

.

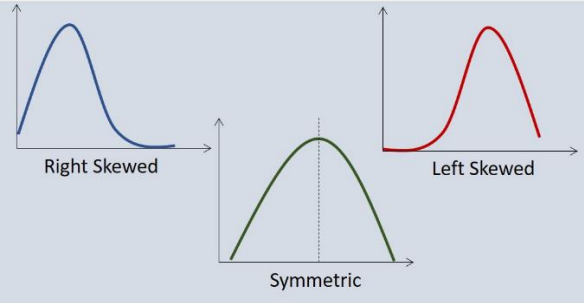

Fig 2. Non-Gaussian/Skewed Distributions

The above visual shows an example of right skewed and left skewed data. The graph in the center is a symmetric and normally distributed data.

Steps for Interquartile Range (IQR) Outlier Detection

Steps:

- **Compute Quartiles**: Determine the first quartile Q1 (25th percentile) and third quartile Q3 (75th percentile) of the variable

- **Calculate** IQR=Q3−Q1
- **Define Outlier Thresholds**: Any data point falling outside the interval:
  [Q1−1.5 × IQR, Q3 + 1.5 × IQR]

is considered an outlier. This threshold may be adjusted (e.g., 2.0 or 3.0 times IQR) depending on the domain-specific tolerance for variability

- Outliers can be handled via capping & flooring or winsorization. This method is particularly effective in skewed financial, demographic, and behavioral datasets where preserving robust central tendency is critical for modeling stability.

Python Implementation [6]:

```
import pandas as pd
# Example data
data = pd.Series([100, 120, 150, 200, 400, 600, 1000, 1500])

# Calculate IQR
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1

# Define bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Cap and floor outliers
data_clipped=data.clip(lower=lower_bound, upper=upper_bound)
```

Median Absolute Deviation (MAD)

The Median Absolute Deviation (MAD**)** is a robust statistical measure of variability, particularly effective in datasets with non-Gaussian distributions and heavy-tailed or contaminated data. Unlike the standard deviation, which is influenced by extreme values, MAD uses the median as a measure of central tendency and is thus resistant to outliers [4]. Median-based statistics, including MAD, are frequently used in robust regression, anomaly detection, and preprocessing pipelines where data quality and distributional assumptions are critical. Because the median is less sensitive to skew and extreme observations, MAD-based detection offers a more stable and interpretable alternative for identifying anomalous points

Steps for MAD – based Outlier Detection:

- Calculate the Median (M): Let 'M' be the median of the data $\{x_1, x_2, x_3 \ldots x_n\}$
- Compute the absolute deviations of each data point from the median. MAD = median $(|x_i - M|)$
- Define outliers using a multiplier k (e.g., 3 or 3.5). A data point x_i is considered an outlier if the values are above or below the given range.
  [M - k. MAD , M + k. MAD]
- While IQR uses percentile cutoffs and is effective in mildly skewed data, MAD is even more robust, especially when the dataset contains **multiple extreme values or irregular distributions**. This makes MAD a preferred method in domains such as fraud detection, healthcare diagnostics, and sensor data analysis where data can be erratic and asymmetric

Python Implementation [6]:

```
from scipy.stats import median_abs_deviation

# Example data
data = np.array([100, 120, 150, 200, 400, 600, 1000, 1500])

# Calculate median and MAD
```

```
median = np.median(data)
mad = median_abs_deviation(data)

# Define bounds
k = 3
lower_bound = median - k * mad
upper_bound = median + k * mad

# Cap and floor outliers
data_clipped = np.clip(data, lower_bound, upper_bound)
```

C. KS Test for Normality Detection

The Kolmogorov–Smirnov (KS) test is a non-parametric statistical test used to assess whether a sample of data originates from a specific theoretical distribution—commonly, the normal distribution. It compares the empirical cumulative distribution function (ECDF) of the observed data with the cumulative distribution function (CDF) of a reference distribution (typically standard normal) and evaluates the maximum distance between the two. This distance is then used to compute a p-value indicating whether the observed differences are statistically significant.

The KS test is particularly useful as a **preprocessing diagnostic** to inform the choice between parametric and non-parametric outlier detection techniques

.

Procedure:

- Define Hypotheses:
  - Null Hypothesis H_0: The data follows a normal distribution.
  - Alternative Hypothesis H_1: The data does not follow a normal distribution.
- Run the KS Test: Apply the KS test to a variable of interest, optionally after standardizing it.
- Interpret the p-value:
  - If $p > 0.05$: Fail to reject H_0 ; the data does not significantly deviate from normality.
  - If $p \leq 0.05$: Reject H_0; the data significantly deviates from a normal distribution

Python Implementation [6]:

```
from scipy.stats import kstest
import numpy as np

# Example data
data = np.random.normal(0, 1, 1000)

# KS Test
ks_stat,ks_p=kstest(data,'norm',args=(np.mean(data), np.std(data)))

if ks_p > 0.05:
   print ("Data is normally distributed.")
else:
   print ("Data is not normally distributed.")
```

Based on the KS test results:

- **Gaussian Distribution**: Use statistical methods like the standard deviation approach.
- **Non-Gaussian Distribution**: Use robust techniques like IQR or MAD

Iterative Refinement

Outlier treatment is often an **iterative process**, especially when working with real-world data that may contain latent skewness, heteroskedasticity, or multi-modality. It is recommended to **reassess the distribution** of each

variable after the initial outlier treatment to ensure that the transformed data adheres to the assumptions required by downstream models.

Steps for Iterative Refinement:

- **Reevaluate Distribution Post-Treatment:**

Reapply normality tests (e.g., KS test, Shapiro–Wilk) to determine whether outlier capping/flooring has stabilized the variable's distribution.

- **Apply Transformations (if needed):**

For variables that remain heavily skewed, consider applying **mathematical transformations** to induce symmetry:

  - **Logarithmic transformation:**
  - **Square root transformation:**
  - **Box–Cox transformation:** A parameterized approach for power transformations.

- **Reassess and Iterate:** Once transformations are applied, retest for normality and repeat the outlier detection process if necessary. This ensures the data is properly prepared for modeling techniques that assume distributional regularity.

D. Domain Specific Limits

In many applied machine learning contexts, particularly in healthcare, automotive, finance, and manufacturing—certain features possess inherent physical or operational constraints. For these variables, relying solely on statistical measures like z-scores, IQR, or MAD may be insufficient or even misleading. Instead, domain expertise can play a crucial role in identifying outliers based on logical boundaries, engineering specifications, or regulatory standards.

Examples of Domain-Specific Constraints:

- **Automotive:** Vehicle mileage should not exceed the plausible upper limit based on age and usage. For example, a newly sold car is unlikely to have over 30,000 miles within the first month.
- **Healthcare:** Body temperature readings beyond 110°F or below 90°F are physiologically implausible and may indicate sensor errors or data entry mistakes.
- **Finance:** Credit utilization rates must fall within [0%, 100%]. Negative values or rates exceeding 100% are typically the result of reporting anomalies.

Steps for Applying Domain-Specific Limits:

- **Consult Subject Matter Experts (SMEs):**
  Gather acceptable value ranges or thresholds for each feature based on historical data, business rules, or industry standards.
- **Flag Logical Violations:** Identify observations that fall outside domain-specified boundaries, regardless of their statistical rarity.
- **Treat Based on Context:** Depending on the nature of the violation:
  - Correct obvious data entry errors (e.g., shifting decimal places).
  - Cap/Floor at allowable boundaries to maintain data integrity.
  - Exclude invalid entries when they cannot be reliably corrected and are known to introduce noise.
- **Document Assumptions:** Clearly annotate and justify domain-based thresholds in data processing documentation to ensure transparency and reproducibility.

Advantages:

- **Improves data realism:** Outliers identified through domain limits often reflect real-world constraints that statistical models cannot infer.
- **Supports model trust:** Ensures that models learn from valid, interpretable patterns, especially in regulated industries.
- **Reduces false positives:** Prevents legitimate extreme values from being incorrectly flagged as anomalies.

Incorporating domain knowledge into outlier detection bridges the gap between statistical rigor and practical application, fostering models that are not only accurate but also contextually grounded and reliable in production.

.

Example:

- For a payment field with a maximum allowable value of $10,000, cap any value exceeding $10,000.

Python Code [6]:

```
# Example data
data = np.array([500, 2000, 5000, 15000, 100000])

# Cap outliers based on domain-specific limits
max_limit = 10000
data_capped = np.clip(data, None, max_limit)
```

E. Influential Rows

Outlier detection not only improves data quality but also enhances the effectiveness of feature selection by increasing signal-to-noise ratio and stabilizing parameter estimates [5]. In regression-based modeling—both linear and logistic—certain observations may exert undue influence on the estimated coefficients. These data points, even if not extreme in terms of input values, can disproportionately affect model predictions and interpretability.

Cook's Distance is a widely used diagnostic measure for detecting such influential observations. It quantifies the extent to which the regression coefficients change when a given data point is removed from the model estimation process.

Threshold for Influence Detection: A common heuristic is to flag observations where $D_i > 4/n$ as potentially influential, where n is the total number of observations in the dataset. Points exceeding this threshold should be carefully examined, as they may indicate data entry errors or valid but rare cases,

Modeling Consideration: Identifying and addressing influential rows using Cook's Distance can lead to:

- Improved model robustness,
- Reduced variance inflation,
- Better generalization on unseen data.

In practice, Cook's Distance is best **used in conjunction with residual plots, leverage statistics, and domain expertise** to make informed decisions about retaining or modifying such observations

Python Implementation [6]:

```
import statsmodels.api as sm

# Example data
X = np.random.rand(100, 2)
y = X[:,0]*10+X[:,1] * 5 + np.random.rand(100)

# Add constant for intercept
X_const = sm.add_constant(X)

# Fit linear regression model
model = sm.OLS(y, X_const).fit()
```

```
# Calculate Cook's Distance
influence = model.get_influence()
cooks_d = influence.cooks_distance[0]

# Identify influential points
threshold = 4 / len(X)
influential_points = np.where(cooks_d > threshold)[0]
```

III. CONCLUSION

Effective outlier treatment directly improves predictive model stability, as supported by Friedman et al. [8]. This paper provides practical techniques and Python code for addressing outliers in Gaussian, non-Gaussian, and domain-specific datasets. Visualization and iterative refinement ensure robust preprocessing, enhancing model accuracy and generalizability.

REFERENCES:

1. J. W. Tukey, *Exploratory Data Analysis*, Reading, MA: Addison-Wesley, 1977.
2. D. C. Montgomery, *Introduction to Statistical Quality Control*, 6th ed., Hoboken, NJ: Wiley, 2009, pp. 110–123..
3. R. A. Barnett and T. Lewis, *Outliers in Statistical Data*, 3rd ed., Wiley, 1994.
4. P. J. Rousseeuw and A. M. Leroy, "Robust regression and outlier detection," *Wiley-Interscience*, vol. 87, pp. 1–256, 1987..
5. H. Liu, and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*, Springer, 1998..
6. J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*, O'Reilly Media, 2016.
7. J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer, 2009.