DATA SCALING & TRANSFORMATION

11 October 2023 21:03

scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.

Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process.

There are several common techniques for feature scaling, including standardization, normalization, and min-max scaling. These methods adjust the feature values while preserving their relative relationships and distributions.

By applying feature scaling, the dataset's features can be transformed to a more consistent scale, making it easier to build accurate and effective machine learning models. Scaling facilitates meaningful comparisons between features, improves model convergence, and prevents certain features from overshadowing others based solely on their magnitude.

Scaling:

- Scaling involves transforming the values of features to a specific range. It maintains the shape of the original distribution while changing the scale.
- Scaling is particularly useful when features have different scales, and certain algorithms are sensitive to the magnitude of the features.
- Common scaling methods include Min-Max scaling and Standardization (Z-score scaling).

Min-Max Scaling:

(Normalization)

Min-max scaling, also known as min-max normalization, is a data normalization technique used to rescale numeric features to a common range.

- Min-Max scaling rescales the values to a specified range, typically between 0 and 1.
- It preserves the original distribution and ensures that the minimum value maps to 0 and the maximum value maps to 1.

m = (x -xmin) / (xmax -xmin)

- Python3

```
from sklearn.preprocessing import MinMaxScaler

# initialising the MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

# Numerical columns
num_col_ = [col for col in X.columns if X[col].dtype != 'object']
x1 = X
# learning the statistical parameters for each of the data and transforming
x1[num_col_] = scaler.fit_transform(x1[num_col_])
x1.head()
```

Standardization (Z-score scaling):

Standard Scaler

- Standardization transforms the values to have a mean of 0 and a standard deviation of 1.
- It centers the data around the mean and scales it based on the standard deviation.
- Standardization makes the data more suitable for algorithms that assume a Gaussian distribution or require features to have zero mean and unit variance.
  $Z = (X - \mu) / \sigma$
  Where,
- X = Data
- μ = Mean value of X
- σ = Standard deviation of X

The choice between normalization and standardization depends on the specific requirements of the dataset and the machine learning algorithm being used. Both techniques have their advantages and are suitable for different scenarios:

1. Normalization:
   - Use normalization when the distribution of the data does not follow a Gaussian distribution and when the scale of the data varies significantly. It is particularly useful when you want to preserve the distribution shape of the data.
   - Normalization is suitable for algorithms that rely on distance measures and where the magnitude of the features is not critical. It helps in preventing features with larger scales from dominating the learning process.

2. Standardization:
   - Use standardization when the data features are normally distributed or when the algorithm assumes that the data is normally distributed. It is effective for transforming data to have a mean of 0 and a standard deviation of 1.
   - Standardization is beneficial when the algorithm is sensitive to the scale of the features, such as in gradient-based optimization algorithms or in algorithms that rely on the Euclidean distance. It helps in ensuring that the features contribute equally to the learning process.

In practice, it is essential to consider the characteristics of the dataset, the requirements of the machine learning algorithm, and the specific objectives of the analysis when deciding whether to use normalization or standardization. Both techniques play a crucial role in preparing the data for machine learning tasks and can significantly impact the performance and convergence of the learning algorithms.

Advantages of Data Cleaning in Machine Learning:

3. Improved model performance: Data cleaning helps improve the performance of the ML model by removing errors, inconsistencies, and irrelevant data, which can help the model to better learn from the data.
4. Increased accuracy: Data cleaning helps ensure that the data is accurate, consistent, and free of errors, which can help improve the accuracy of the ML model.
5. Better representation of the data: Data cleaning allows the data to be transformed into a format that better represents the underlying relationships and patterns in the data, making it easier for the ML model to learn from the data.
6. Improved data quality: Data cleaning helps to improve the quality of the data, making it more reliable and accurate. This ensures that the machine learning models are trained on high-quality data, which can lead to better predictions and outcomes.
7. Improved data security: Data cleaning can help to identify and remove sensitive or confidential information that could compromise data security. By eliminating this information, data cleaning can help to ensure that only the necessary and relevant data is used for machine learning.

Disadvantages of Data Cleaning in Machine Learning:

8. Time-consuming: Data cleaning can be a time-consuming task, especially for large and complex

datasets.

9. Error-prone: Data cleaning can be error-prone, as it involves transforming and cleaning the data, which can result in the loss of important information or the introduction of new errors.
10. Limited understanding of the data: Data cleaning can lead to a limited understanding of the data, as the transformed data may not be representative of the underlying relationships and patterns in the data.
11. Data loss: Data cleaning can result in the loss of important information that may be valuable for machine learning analysis. In some cases, data cleaning may result in the removal of data that appears to be irrelevant or inconsistent, but which may contain valuable insights or patterns.
12. Cost and resource-intensive: Data cleaning can be a resource-intensive process that requires significant time, effort, and expertise. It can also require the use of specialized software tools, which can add to the cost and complexity of data cleaning.
13. Overfitting: Overfitting occurs when a machine learning model is trained too closely on a particular dataset, resulting in poor performance when applied to new or different data. Data cleaning can inadvertently contribute to overfitting by removing too much data, leading to a loss of information that could be important for model training and performance.

DATA TRANSFORMATION:

Data transformation refers to the process of converting or altering data from its original format into a more suitable format for analysis, visualization, or other data-related tasks. It involves modifying the structure, format, or values of the data to make it more compatible with the requirements of specific analytical techniques or applications.

Robust Scaler

Robust Scaler are robust to outliers.It is used to scale the feature to median and quantiles Scaling using median and quantiles consists of substracting the median to all the observations, and then dividing by the interquantile difference. The interquantile difference is the difference between the 75th and 25th quantile:

IQR = 75th quantile - 25th quantile

X_scaled = (X - X.median) / IQR

The RobustScaler is a data transformation technique used in machine learning to scale features to median and quantiles, which makes it more robust to outliers compared to standard scaling techniques like the StandardScaler.

Here are some reasons why the RobustScaler is used:

1. Outlier Sensitivity: The RobustScaler is less sensitive to outliers compared to standard scaling techniques. **It uses the median and interquartile range, which makes it robust to data points that deviate significantly from the rest of the data.**

2. Preservation of Scale: The RobustScaler preserves the scale of the data, **ensuring that the central tendency and spread of the features are maintained while minimizing the influence of outliers.**

3. Non-Normal Data: It is effective for data that does not follow a normal distribution or has a skewed distribution, as it uses the median and quartiles for scaling, which are more robust to deviations from the normal distribution.

4. Improved Model Performance: By using the RobustScaler, you can improve the performance of machine learning models that are sensitive to outliers, ensuring that the presence of outliers does not disproportionately affect the model training process.

The RobustScaler is a valuable tool in data preprocessing, especially when dealing with datasets that contain outliers or have non-normal distributions.

Using the RobustScaler on data that looks pretty normal**(normally distributed)** without any strange or really big or small numbers doesn't change the data much. It's like applying a special tool to something that's already working fine. The RobustScaler is really meant for data that has unusual or extreme values, so if your data doesn't have those, using the RobustScaler might not do anything noticeable. It's like trying to fix something that isn't broken.
the RobustScaler may not significantly alter the statistical properties of the data, as the median and interquartile

range would be relatively similar to the mean and standard deviation in a normal distribution.

Guassian Transformation:

Some machine learning algorithms like linear and logistic assume that the features are normally distributed -Accuracy -Performance

- logarithmic transformation
- square root transformation
- exponential transformation (more general, you can use any exponent)
- boxcox transformation

logarithmic transformation

The log transformation is, arguably, the most popular among the different types of transformations used to transform skewed data to approximately conform to normality.

If the original data follows a log-normal distribution or approximately so, then the log-transformed data follows a normal or near normal distribution.

A logarithmic transformation is a data transformation method that involves applying the logarithm function to each data point in a given dataset. It is commonly used to stabilize variance, make the data more symmetric, or linearize the relationship between variables.
The logarithmic function is the inverse of an exponential function and is defined as follows:

$y=\log_b(x)$

To determine whether your data follows an exponential trend and whether a logarithmic transformation is appropriate, you can visually inspect the data and analyze its behavior.

Statistical Tests: Conduct tests for normality, such as the **Shapiro-Wilk test or visual assessments like Q-Q plots**, to evaluate whether the data significantly deviates from a normal distribution. If the data is skewed, implementing a logarithmic transformation might help in achieving symmetry.

Trial and Error: Try applying the logarithmic transformation to the data and assess the impact on the distribution and relationships within the dataset. Compare the results before and after the transformation to gauge its effectiveness.

HOW TO APPLY LOG TRANSFORMATION FOR NEGATIVE VALUES?

Applying a logarithmic transformation to a dataset that contains negative values directly is not possible because the logarithm function is not defined for negative numbers in the real number system. The logarithm of a negative number is considered a complex number in mathematics.

However, if you have a dataset that includes both positive and negative values, you might consider using a different transformation approach, such as the Box-Cox transformation, which can handle both positive and negative values. The Box-Cox transformation is a power transformation that can be applied to stabilize variance and make the data more normally distributed.

Alternatively, if your dataset contains a mixture of positive and negative values and you still want to apply a logarithmic transformation, you might consider shifting the values to make them all positive before applying the transformation. One approach is to add a constant value to all the data points to ensure that they are non-negative, and then apply the logarithmic transformation. However, it's important to interpret the results of such a transformation with caution and consider the context of the data and the implications of the manipulation.

- ## Square root transformation

square root transformation is a mathematical operation applied to data to make it more suitable for

analysis or modelling. It's a type of data transformation used to make the data more closely resemble a normal distribution or to stabilize the variance of the data. Here's how it works in simple terms:

Square Root Transformation: For each data point in your dataset, you take the square root of that value. So, if you have a number like 25, the square root of 25 is 5, and that becomes the new value for that data point.
Example:
Suppose you have a dataset of the ages of a group of people: [9, 16, 25, 36, 49]. If you apply a square root transformation to this data, it would become: [3, 4, 5, 6, 7].

- ## Why Use Square Root Transformation:

A square root transformation is typically applied to data to stabilize variance, make the data less skewed, or linearize relationships in cases where the data exhibits a right-skewed distribution.

Stabilizing Variance: In some statistical models, it's assumed that the variance of the data is constant. If your data has increasing variance with increasing values, a square root transformation can help stabilize it.

Normalization: It can help make the data more normally distributed, which is often an assumption in statistical tests and models. Normal distribution is a bell-shaped curve, and square root transformation can make your data more bell-shaped.

Interpretability: In some cases, taking the square root of values can make them more interpretable. For example, if you're dealing with areas, like square feet, the square root can transform them into lengths, which might be easier to understand.

CASES WHERE SQUARE ROOT TO BE CONSIDERED:

1. **Variance Stabilization**: If the variance of the data increases with the mean, applying a square root transformation can help stabilize the variance, making it more consistent across different levels of the data.

2. **Skewness Reduction:** When the data is right-skewed, a square root transformation can help reduce the skewness and make the data more symmetric, enabling the application of statistical methods that assume normality.

3. **Linearization of Relationships:** If you suspect that the relationship between variables is nonlinear and closer to a square root function, applying a square root transformation can help linearize the relationship and make it more amenable to linear modelling techniques.

How do we apply square root transformation for negative values.?

The square root transformation is not directly applicable to negative values because the square root of a negative number is not defined in the real number system. However, if you have a dataset that contains both positive and negative values and you want to apply a transformation that can handle them, you might consider using a different approach, such as the Yeo-Johnson transformation.

The Yeo-Johnson transformation is an extension of the Box-Cox transformation and can handle data with both positive and negative values. It is defined as:

$$y = \begin{cases} (x+1)^{\lambda} & \text{if } x \geq 0, \lambda \neq 0 \\ -((-x+1)^{2-\lambda} - 1) & \text{if } x < 0, \lambda \neq 2 \\ \ln(x+1) & \text{if } x \geq 0, \lambda = 0 \\ -\ln(-x+1) & \text{if } x < 0, \lambda = 2 \end{cases}$$

```python
import numpy as np
from scipy.stats import yeojohnson

# Example dataset with both positive and negative values
data = np.array([-2, -1, 0, 1, 2])

# Apply the Yeo-Johnson transformation
transformed_data, lambda_ = yeojohnson(data)

# Print the transformed data and the lambda value
print("Transformed Data:", transformed_data)
print("Lambda Value:", lambda_)
```

The output of the provided code snippet, when executed, might look similar to the following:

```yaml
Transformed Data: [       nan        nan 0.         1.33114906 1.86120972]
Lambda Value: 0.30151134457776363
```

The transformed data array represents the Yeo-Johnson transformation applied to the original data. The `nan` values in the transformed data correspond to the negative values in the original data since the Yeo-Johnson transformation does not handle negative values for certain lambda values. The lambda value represents the parameter estimated during the transformation process and is essential for the inverse transformation, if necessary.

If you have specific data to transform, replace the `data` array with your dataset, ensuring that it contains both positive and negative values for the Yeo-Johnson transformation to be applicable.

- **exponential transformation**

An exponential transformation in data science involves raising data values to an exponent. This transformation is useful for handling data that grows or decays at a consistent rate. In simple terms, it makes the data grow or shrink much faster.
Exponential Transformation Example:

Imagine you have a dataset representing the population of a town over several years:

- Year 1: 1,000
- Year 2: 1,100
- Year 3: 1,210
- Year 4: 1,331
- Year 5: 1,464

If you apply an exponential transformation with an exponent of 2, it would look like this:

- Year 1: 1,000^2 = 1,000,000
- Year 2: 1,100^2 = 1,210,000
- Year 3: 1,210^2 = 1,464,100
- Year 4: 1,331^2 = 1,772,161
- Year 5: 1,464^2 = 2,144,896

In this example, the exponential transformation squared the values, making the growth more pronounced.

Why Use Exponential Transformation:

Highlighting Trends: Exponential transformations can help emphasize trends or patterns that might not be as apparent in the original data. For example, in finance, exponential growth is often used to model compound interest or investment returns.

Stabilizing Variance: It can help stabilize the variance in data, just like other transformations. If your data has a variance that increases or decreases with the magnitude of values, an exponential transformation can mitigate this effect.

Applying Linear Models: In some cases, you might want to use linear models to analyse data that exhibits exponential growth. Applying an exponential transformation can make the data more suitable for linear regression analysis.

Addressing Heteroscedasticity: In regression analysis, if the residuals display a pattern where the variance of the residuals changes across the range of predicted values, applying an exponential transformation to the response variable can help mitigate heteroscedasticity.

Applying an exponential transformation directly to a dataset that contains negative values can pose challenges because the exponential function is typically defined for non-negative numbers. However, if you have a dataset that includes both positive and negative values and you wish to transform it using an exponential function, you might consider applying an offset to the data before transformation.
One way to handle negative values is to add a constant to all values to ensure that they are non-negative before applying the exponential transformation. This constant can be chosen based on the magnitude of the negative values and the context of the data. After adding the offset, you can apply the exponential transformation to the adjusted data.

Boxcox transformation:

The Box-Cox transformation is a method developed by George Box and Sir David Cox to transform **non-normal or skewed data into a more normally distributed form.**
It involves applying a power function to the original data, which helps address issues of heteroscedasticity and improve the assumptions of normality and constant variance in statistical models.

Certainly! The Box-Cox transformation formula is Y(lambda) = (X^lambda - 1) / lambda.

In this formula, **Y(lambda) represents the transformed data**, X is the original data, and lambda is the transformation parameter.

By varying the value of lambda, different types of transformations can be applied to the data. The optimal lambda value is determined using statistical techniques like maximum likelihood estimation.

BENEFITS:
The Box-Cox transformation is valuable because it helps normalize skewed data and improves the assumptions required by many statistical techniques, such as linear regression. It can address issues of non-normality and heteroscedasticity, allowing for more accurate and reliable statistical modeling. By transforming the data, it helps meet the assumptions of normality and constant variance, enhancing the validity of the results.

EXAMPLE:
Let's say we're analyzing a dataset where the target variable is right-skewed, meaning it has a long tail on the right side. By applying the Box-Cox transformation to the target variable, we can bring it closer to a normal distribution, which can improve the performance of statistical models that assume normality.

This transformation can be particularly beneficial in fields like finance, economics, and social sciences, where data often deviates from normality.