Feature Engineering: Scaling, Normalization, and Standardization

If you are an ML practitioner then you must have come across the term feature scaling which is considered as an unskippable part of the data processing cycle so, that we can achieve stable and fast training of our ML algorithm. In this article, we will learn about different techniques which are used to perform feature scaling in practice.

What is Feature Scaling?

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

Why use Feature Scaling?

In machine learning, feature scaling is employed for a number of purposes:

- Scaling guarantees that all features are on a comparable scale and have comparable ranges. This process is known as feature normalisation. This is significant because the magnitude of the features has an impact on many machine learning techniques. Larger scale features may dominate the learning process and have an excessive impact on the outcomes. You can avoid this problem and make sure that each feature contributes equally to the learning process by scaling the features.
- Algorithm performance improvement: When the features are scaled, several machine learning methods, including gradient descent-based algorithms, distance-based algorithms (such k-nearest neighbours), and support vector machines, perform better or converge more quickly. The algorithm's performance can be enhanced by scaling the features, which can hasten the convergence of the algorithm to the ideal outcome.
- Preventing numerical instability: Numerical instability can be prevented by avoiding significant scale disparities between features. Examples include distance calculations or matrix operations, where having features with radically differing scales can result in numerical overflow or underflow problems. Stable computations are ensured and these issues are mitigated by scaling the features.
- Scaling features makes ensuring that each characteristic is given the same consideration during the learning process. Without scaling, bigger scale features could dominate the learning,

producing skewed outcomes. This bias is removed through scaling, which also guarantees that each feature contributes fairly to model predictions.

Absolute Maximum Scaling

This method of scaling requires two-step:

1. We should first select the maximum absolute value out of all the entries of a particular measure.
2. Then after this, we divide each entry of the column by this maximum value.

$$X_{\text{scaled}} = \frac{X_i - \max(|\mathrm{X}|)}{\max(|\mathrm{X}|)}$$

After performing the above-mentioned two steps we will observe that each entry of the column lies in the range of -1 to 1. But this method is not used that often the reason behind this is that it is too sensitive to the outliers. And while dealing with the real-world data presence of outliers is a very common thing.

For the demonstration purpose, we will use the dataset which you can download from here. This dataset is a simpler version of the original house price prediction dataset having only two columns from the original dataset. The first five rows of the original data are shown below:

Python3

```
import  pandas as pd

df  =  pd.read_csv(``'SampleFile.csv'``)

print``(df.head())
```

Output:

```
   LotArea  MSSubClass
0     8450          60
1     9600          20
2    11250          60
3     9550          70
4    14260          60
```

Now let’s apply the first method which is of the absolute maximum scaling. For this first, we are supposed to evaluate the absolute maximum values of the columns.

Python3

```
import  numpy as np

max_vals  =  np.``max``(np.``abs``(df))

max_vals
```

Output:

```
LotArea       215245
MSSubClass       190
dtype: int64
```

Now we are supposed to subtract these values from the data and then divide the results from the maximum values as well.

Python3

```
print``((df  -  max_vals)  /  max_vals)
```

Output:

```
       LotArea  MSSubClass
0    -0.960742   -0.684211
1    -0.955400   -0.894737
2    -0.947734   -0.684211
3    -0.955632   -0.631579
4    -0.933750   -0.684211
...        ...         ...
1455 -0.963219   -0.684211
1456 -0.938791   -0.894737
1457 -0.957992   -0.631579
1458 -0.954856   -0.894737
1459 -0.953834   -0.894737
[1460 rows x 2 columns]
```

Min-Max Scaling

This method of scaling requires below two-step:

1. First, we are supposed to find the minimum and the maximum value of the column.

2. Then we will subtract the minimum value from the entry and divide the result by the difference between the maximum and the minimum value.

$$X_{\text{scaled}} = \frac{X_i - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

As we are using the maximum and the minimum value this method is also prone to outliers but the range in which the data will range after performing the above two steps is between 0 to 1.

Python3

```
from  sklearn.preprocessing  import  MinMaxScaler

scaler  =  MinMaxScaler()

scaled_data  =  scaler.fit_transform(df)

scaled_df  =  pd.DataFrame(scaled_data,

                  columns``=``df.columns)

scaled_df.head()
```

Output:

```
     LotArea  MSSubClass
 0  0.033420    0.235294
 1  0.038795    0.000000
 2  0.046507    0.235294
 3  0.038561    0.294118
 4  0.060576    0.235294
```

Normalization

This method is more or less the same as the previous method but here instead of the minimum value, we subtract each entry by the mean value of the whole data and then divide the results by the difference between the minimum and the maximum value.

$$X_{\text{scaled}} = \frac{X_i - X_{\text{mean}}}{X_{\text{max}} - X_{\text{min}}}$$

Python3

```
from  sklearn.preprocessing  import  Normalizer

scaler  =  Normalizer()

scaled_data  =  scaler.fit_transform(df)

scaled_df  =  pd.DataFrame(scaled_data,

                 columns``=``df.columns)

print``(scaled_df.head())
```

Output:

```
    LotArea  MSSubClass
0  0.999975    0.007100
1  0.999998    0.002083
2  0.999986    0.005333
3  0.999973    0.007330
4  0.999991    0.004208
```

Standardization

This method of scaling is basically based on the central tendencies and variance of the data.

1. First, we should calculate the mean and standard deviation of the data we would like to normalize.
2. Then we are supposed to subtract the mean value from each entry and then divide the result by the standard deviation.

This helps us achieve a normal distribution(if it is already normal but skewed) of the data with a mean equal to zero and a standard deviation equal to 1.

$$X_{\text{scaled}} = \frac{X_i - X_{\text{mean}}}{\sigma}$$

Python3

```
from  sklearn.preprocessing  import  StandardScaler

scaler  =  StandardScaler()

scaled_data  =  scaler.fit_transform(df)
```

```
scaled_df  =  pd.DataFrame(scaled_data,

               columns``=``df.columns)

print``(scaled_df.head())
```

Output:

```
     LotArea  MSSubClass
0 -0.207142    0.073375
1 -0.091886   -0.872563
2  0.073480    0.073375
3 -0.096897    0.309859
4  0.375148    0.073375
```

Robust Scaling

In this method of scaling, we use two main statistical measures of the data.

- Median
- Inter-Quartile Range

After calculating these two values we are supposed to subtract the median from each entry and then divide the result by the interquartile range.

$$X_{\text{scaled}} = \frac{X_i - X_{\text{median}}}{IQR}$$

Python3

```
from  sklearn.preprocessing  import  RobustScaler

scaler  =  RobustScaler()

scaled_data  =  scaler.fit_transform(df)

scaled_df  =  pd.DataFrame(scaled_data,

               columns``=``df.columns)

print``(scaled_df.head())
```

Output:

```
    LotArea  MSSubClass
0 -0.254076         0.2
1  0.030015        -0.6
2  0.437624         0.2
3  0.017663         0.4
4  1.181201         0.2
```