Introduction to Feature Scaling: Normalization and Standardization

Big data science and machine learning organizations record many attributes/properties to avoid losing critical information. Each attribute has its own properties and valid ranges in which it can fall. For example, the speed of a motorcycle can be in the range of 0-200 KM/h, while the speed of a car can be in the range of 0-400 KM/h. Machine learning or deep learning models expect these ranges to be on the same scale in order to determine the importance of these properties without bias.

In this article, we will explore one of the important topics used in scaling different attributes for machine learning: **normalization** and **standardization**. There is often confusion among machine learning professionals about which one to use. We will attempt to clear up this confusion in this article.

Key takeaways from this article

- What is Normalization?
- Why do we need scaling (normalization or standardization)?
- What are different normalization techniques?
- What is Standardization?
- When to normalize and when to standardize?

Let's start with the fundamental question.

What is Normalization?

In machine learning, a feature is a measurable property or characteristic of an observed phenomenon that is used as input to train a model. For example, when training a model to predict the price of a flat, we might use the size of the flat as a feature. Adding the locality of the flat as an additional feature can improve the model's performance.

Normalization is the process of scaling all features to the same definite range. It is important to normalize the features because the numerical values of different features may vary greatly. If the ranges of the values of different features are significantly different, then some features may dominate the training process, leading to poor model performance. Normalization helps to ensure that all features are given equal weight and have an equal influence on the model.

Why scale the features?

Let’s go through one example to answer this question, which will open the mathematical angle supporting Normalization or Standardization.

Mathematical Intuition

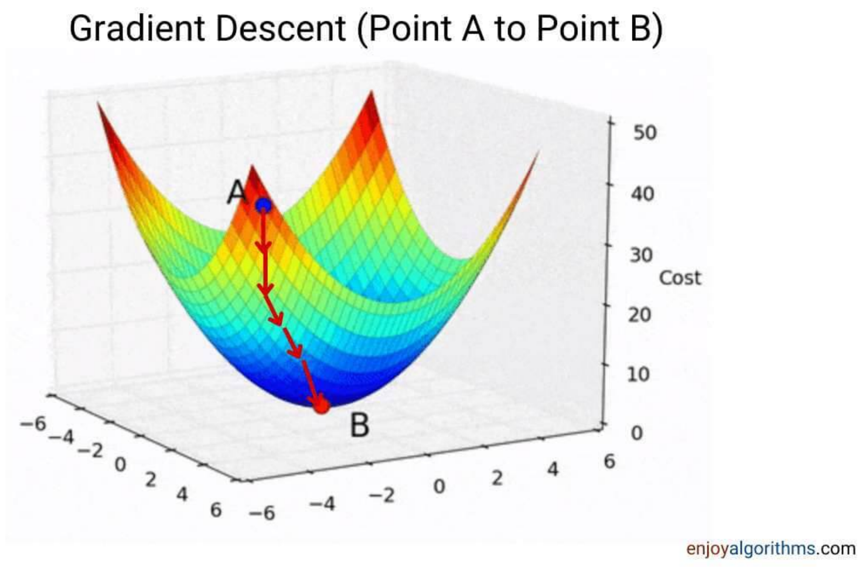
Suppose we make a machine learning model to learn the function Y = **θ1***X + **θ0**. We can use a supervised dataset, which includes input values (X) and corresponding output values (Y), to train a machine learning model. During the training process, the model will start with initial values for certain parameters (such as θ1 and θ0). These initial values may be randomly selected or manually set.

The model will then iteratively adjust these parameters in order to minimize the error between the predicted output values (Y^) and the actual output values (Y). The error is typically measured using a cost function, such as Mean Squared Error (MSE). Our goal is to find values for the parameters that minimize the error and improve the model's performance.

Let’s choose Mean Squared Error (MSE), our error function, also called a cost function. The formulae for MSE are given in the below equation, where **n** is the number of training samples.

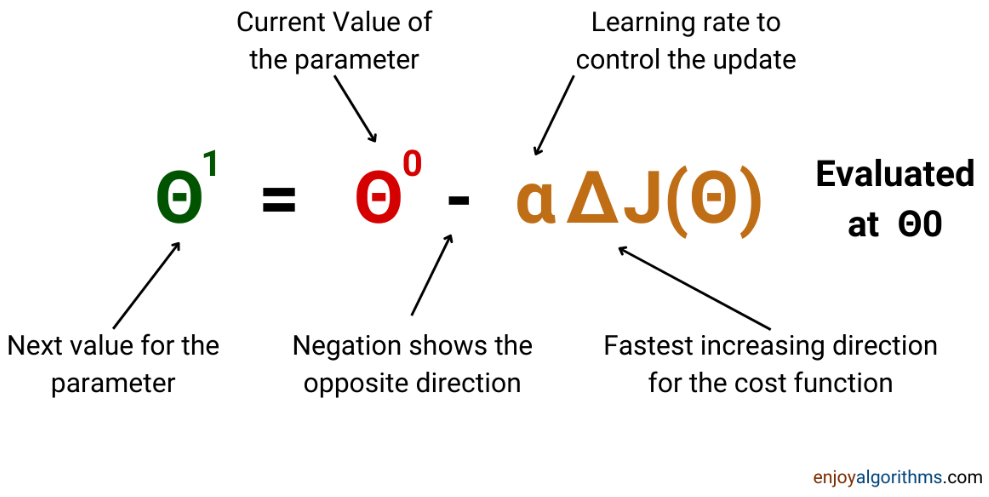
```
       1
MSE = --- * Σ (Y - Ŷ)^2,
       N

 (Y - Ŷ)^2 --> The Square of the difference between actual and predicted v
```

As Y is a function that depends upon two variables, **θ1,** and **θ0, hence** cost function will also depend on these two variables. In the GIF below, there is one dimension of the Cost function, and the rest two dimensions can be considered as **θ1** and **θ0.**



At the start, suppose we are at position A (Shown in GIF above), and reaching position B is our ultimate goal as that is the minimum of the cost function. For that, the machine will tweak the values of **θ1** and **θ0.**

But the machine can take infinite values for **θ1** and **θ0** if it randomly selects them at each step. We use optimizers to help the machine choose the following values of **θ1** and **θ0 to reach** the minima quickly. Let's choose **gradient descent** as our optimizer to learn the function **Y = θ1*X + θ0**. In gradient descent, we update any parameter value using the formulae below.


Current Value of the parameter
Learning rate to control the update
$\Theta^1 = \Theta^0 - \alpha \Delta J(\Theta)$
Evaluated at Θ0
Next value for the parameter
Negation shows the opposite direction
Fastest increasing direction for the cost function
enjoyalgorithms.com

Let's say we updated the value of **θ1** and **θ0** by using the above formulae. Then new values will be:

```
                   ∂ J(θ0, θ1)
θ0 :=  θ0  -  ---------------, θ1 will be treated as constant
                     ∂ θ0

                   ∂ J(θ0, θ1)
θ1 :=  θ1  -  ---------------, θ0 will be treated as constant
                     ∂ θ1
```

Let's calculate `` `δJ(θ0, θ1))/δθ1` `` and `` `δJ(θ0, θ1)/δθ0` ``. Prediction error can be represented in the equation as **error = (Ŷ — Y)**

Cost function :

```
                1    N
J(θ0, θ1)  = --- * Σ (Yi-Ŷi)^2
                N    i
```

Now let's calculate the partial derivative of this cost function concerning two variables, **θ1** and **θ0.**

$$\frac{\partial J(\theta_1)}{\partial \theta_1} = \frac{1}{n}\Sigma_i^n[2 * error * \frac{\partial(error_i)}{\partial \theta_1}]$$

$$\frac{\partial J(\theta_0)}{\partial \theta_0} = \frac{1}{n}\Sigma_i^n[2 * error * \frac{\partial(error_i)}{\partial \theta_0}]$$

Also,

$$\frac{\partial(error)_i}{\partial \theta} = \frac{\partial(\hat{Y} - Y)_i}{\partial \theta},$$

$$\hat{Y} = \theta_1 * X + \theta_0$$

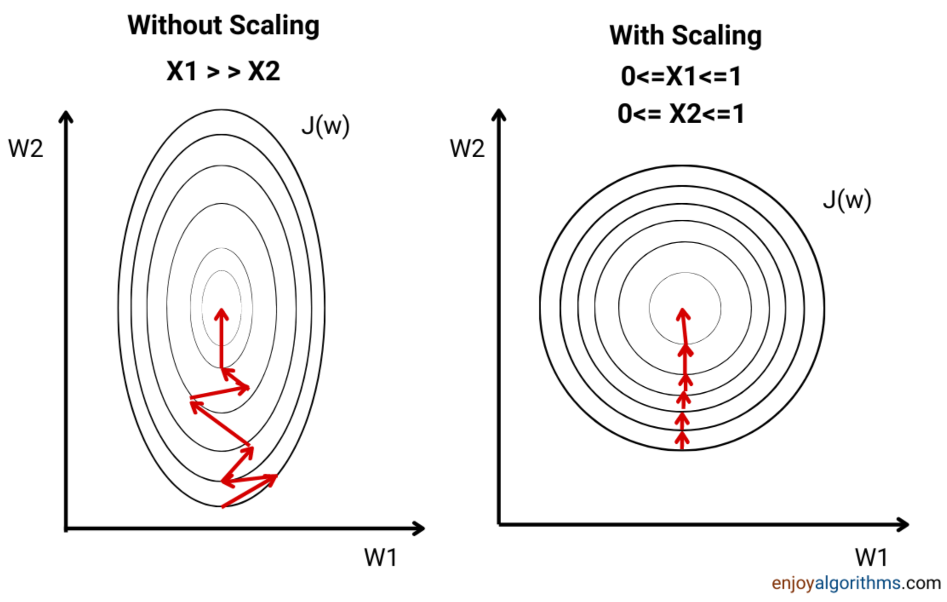
$$\frac{\partial(\hat{Y} - Y)_i}{\partial \theta_1} = \frac{\partial(\theta_1 * X_i + \theta_0 - Y_i)}{\partial \theta_1} = X_i$$

$$\frac{\partial(\hat{Y} - Y)_i}{\partial \theta_0} = \frac{\partial(\theta_1 * X_i + \theta_0 - Y_i)}{\partial \theta_0} = 1$$

After combining the equations and putting everything in the gradient descent formulae,

```
θ1 = θ1 - α * (2*error*X)

θ0 = θ0 - α * (2*error)
```

The **feature value X** in the above update formula will affect the **step size of the gradient descent**. If the features are in different ranges, it will cause different step sizes for every feature. In the image below, X1 and X2 are two attributes that constitute the input variable X, i.e., X = [X1, X2]. Consider this X1 and X2 as two dimensions. To ensure the functionality of the gradient descent moves smoothly towards the minima and steps for gradient descent get updated at the same rate in every dimension, we scale the data before feeding it to the model.



Example Intuition

Some machine learning algorithms are susceptible to Normalization or Standardization, and some are insensitive. Algorithms like **SVM, K-NN, K-means, Neural Networks, or Deep-learning** are susceptible to Normalization/Standardization. These algorithms use the spatial relationships (Space dependent relations) present among the data samples. Let's take an example,

Students	English marks out of 10	Maths marks out of 100
A	5	90
B	7	95
C	9	85
D	10	90

E	8	92

```
Distance (A , B) = √((5-7)^2 + (90-95)^2) = 5.385
Distance (B , C) = √((7-9)^2 + (95-85)^2) = 10.198
```

Let’s use the scaling technique and the percentage of marks instead of direct marks.

Students	English marks in percent	Maths marks in percent
A	0.5	0.9
B	0.7	0.95
C	0.9	0.85
D	1.0	0.9
E	0.8	0.92

```
Distance (A , B) = √((0.5-0.7)^2 + (0.9-0.95)^2) = 0.2061
Distance (B , C) = √((0.7-0.9)^2 + (0.95-0.85)^2) = 0.2236
```

The scaled distances are closer and can be compared easily.

Algorithms like **Decision trees, Random forests, or other tree-based algorithms** are insensitive to Normalization or Standardization as they are applied to every feature individually and are not influenced by any other feature.

Two reasons that support the need for scaling are:

Scaling the features in a machine learning model can improve the optimization process by making the flow of gradient descent smoother and helping algorithms reach the minimum of the cost function more quickly.

Without scaling features, the algorithm may be biased toward the feature with values higher in magnitude. Hence we scale features that bring every feature in the same range, and the model uses every feature wisely.

We know why scaling, so let’s see some popular techniques used to scale all the features in the same range.

Popular Scaling techniques

Min-Max Normalization

This is the most used normalization technique in the machine learning industry. We bring every attribute in the defined range, starting from **a** and ending at **b**. We map every feature's minimum and maximum value to the starting and ending range values. Range values [0, 1] and [-1, 1] are the most popular ones. Formula all three cases are given below:

```
             Xi - mininum(X)
Xi =   ---------------------------,            if the defined range is [0,
         maximum(X) - mininum(X)

             Xi - mininum(X)
Xi = 2* --------------------------  - 1,      if the defined range is [-1,
          maximum(X) - mininum(X)

                 Xi - mininum(X)
Xi = (b-a)* --------------------------  + a,   if the defined range is [a,
              maximum(X) - mininum(X)
```

We generally do not implement these scaling techniques from scratch and use inbuilt functions from Scikit-learn:

```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit_transform(data))

'''
[[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
'''
```

For a custom range, we can use `` `scaler = MinMaxScaler(feature_range=(a, b)).` ``

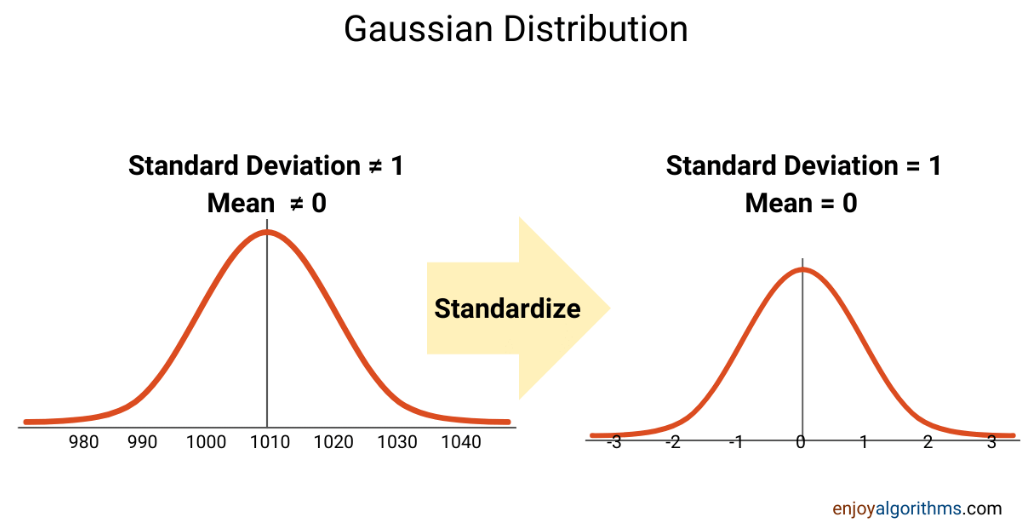
Logistic Normalization

Here, we transform the features to start contributing proportionally in the updation step of gradient descent. If we see the formula below, we exponentially raise the value of X.

```
              1
Xi = -----------------
       1 + exp(-Xi)
```

Standardization

Standardization is another scaling technique in which we transform the feature such that the **changed features** will have **mean (μ) = 0** and **standard deviation (σ) = 1.**



The formula to standardize the features in data samples is :

```
        Xi - μ
Xi = -----------
          Xi
```

This scaling technique is also known as **Z-Score normalization or Z-mean normalization**. Unlike Normalization, Standardization techniques are not much affected by the presence of outliers (**Think how!**).

We generally use the inbuilt function `` `StandardScaler` `` from scikit-learn to standardize the data like this:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
```

```
print(scaler.fit_transform(data))

'''
[[-1.18321596 -1.18321596]
 [-0.50709255 -0.50709255]
 [ 0.16903085  0.16903085]
 [ 1.52127766  1.52127766]]
'''
```

In normalization, we have min and max operations in the formula. What if some outliers have significantly higher/lower magnitudes? They will affect the calculations, and hence normalization is much influenced by the presence of outliers. But if the data samples are large and there are some outliers, then the mean and standard deviation calculations will be affected by a smaller margin. Hence Standardization is less affected by outliers.

Now, we know two different scaling techniques. But sometimes, knowing more or having more options brings another challenge of **choice.** So we have a new question for us,

When to Normalize and When to Standardize?

Let's list the use cases where Normalization and Standardization would benefit.

Normalization would be more beneficial when:

- Data samples are **NOT** normally distributed.
- Dataset is clean or free from outliers.
- The dataset covers all the corner (Minimum or Maximum) ranges of features.
- They are often used for the algorithms like Neural Networks, K-NN, and K-means.

Standardization would be more beneficial when:

- Data samples are from a normal distribution. This is sometimes true, but most effectiveness will be observed when it happens.
- The dataset contains outliers that can affect the min/max calculations.

Summary

Scaling features helps optimization algorithms to reach the minima of cost function quickly.

Scaling features restrict models from being biased towards features having higher/lower magnitude values.

Normalization and Standardization are two scaling techniques.

With gaussian(normal) distributed data samples, Standardization works perfectly.

Possible Interview Questions

1. What is data normalization and why do we need it?
2. Do we need to normalize the output/target variable as well?
3. What is standardization? When is standardization preferred?
4. Why will the model become biased if we do not scale the variables?
5. Why is standardization better in real-life scenarios?

Conclusion

In this article, we discussed the importance of scaling the features in a machine learning model. It is important to scale the features so that they are all on the same scale, as this helps the model to assign equal importance to all features and make predictions without bias. We provided two examples to demonstrate how scaling can improve the performance of a machine learning model. We also mentioned that scaling techniques can be challenging, even for experienced machine learning professionals. We hope you found this article helpful.