Transforming Features to fit Normal Distribution

medium.com/@singh-jasraj/transforming-features-to-fit-normal-distribution-ba1ef51fe860

Jasraj Singh May 27, 2022

Jasraj Singh

Gaussian distribution plays a central role in Machine Learning, with several methods assuming univariate or multivariate normal distributions. Gaussian assumption makes it easier to establish theoretical results.

As explains in his blog:

> A large portion of the field of statistics is concerned with methods that assume a Gaussian distribution: the familiar bell curve.
>
> If your data has a Gaussian distribution, the parametric methods are powerful and well understood. This gives some incentive to use them if possible. Even if your data does not have a Gaussian distribution.

```
standard_normal = np.random.normal(0, 1, size=1_000_000)fontdict =
{'family':'serif', 'color':'darkgreen', 'size':16}fig, axs = plt.subplots(1, 1,
figsize=(8, 8))axs.hist(standard_normal, bins=1000, density=True, fc=
(0,0,1,0.4))axs.set_title('Standard Normal Distribution', fontdict=fontdict,
fontweight='bold', pad=12)axs.set_xlabel('X', fontdict=fontdict,
fontweight='normal', labelpad=12)axs.set_ylabel('Density', fontdict=fontdict,
fontweight='normal', labelpad=12)axs.grid()plt.savefig('random_nomral.jpg')
```

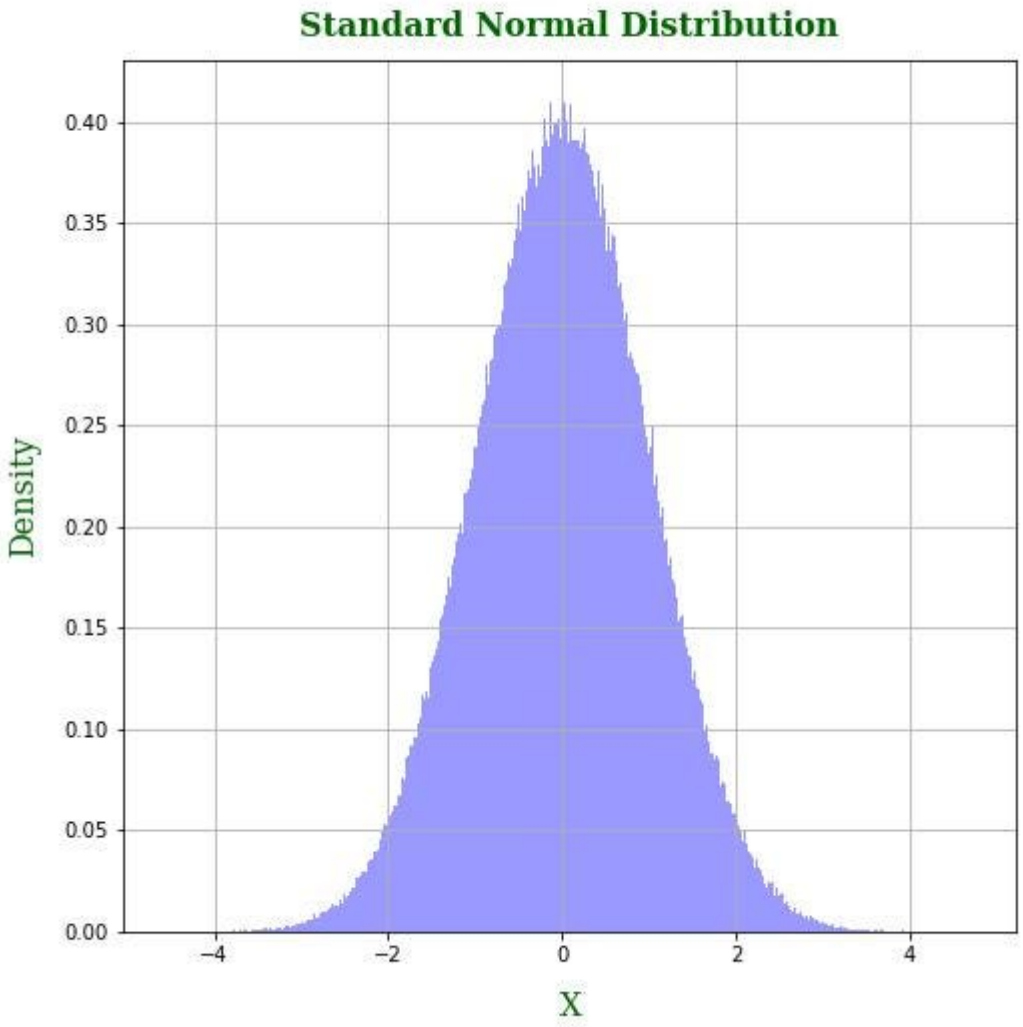

Figure 1: Standard Normal Distribution

Why am I writing an article about this?

For my current internship at Shopee, Singapore, I was working with a feature which had a (approximately) linearly decreasing density (see Figure 2).

```
x = np.linspace(0, 1, 1001)sample = (3 - np.sqrt(9 - 8 * np.random.uniform(0, 1, 1_000_000))) / 2fontdict = {'family':'serif', 'color':'darkgreen', 'size':16}fig, axs = plt.subplots(1, 1, figsize=(8, 8))axs.hist(sample, bins=1000, density=True, fc=(0,0,1,0.4))axs.scatter(x, np.full_like(x, 0.01), c=x, cmap=cmap)axs.set_title('Original Feature Distribution', fontdict=fontdict, fontweight='bold', pad=12)axs.set_xlabel('X', fontdict=fontdict, fontweight='normal', labelpad=12)axs.set_ylabel('Density', fontdict=fontdict, fontweight='normal', labelpad=12)axs.grid()
```

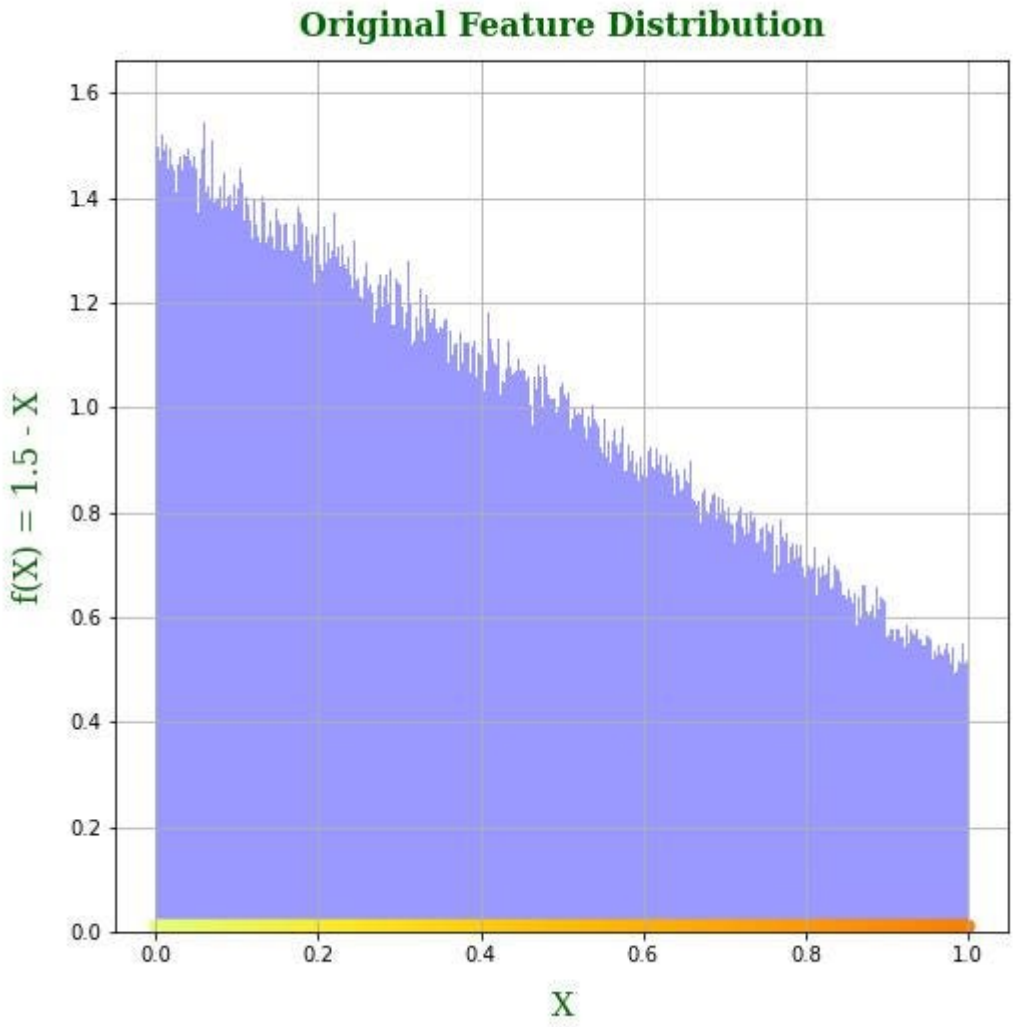

Figure 2: Density Plot for the Feature

I wanted to transform this variable into one with a bell shaped distribution. I talked to a colleague about it, and he said we cannot do much about it (in his defense, the density was not nearly as *nice* as in Figure 2). “Hmm, that seems wrong”, I thought. Surely there must be some transformation that could give me a nice distribution. What if I use some transform that would throw the left end, where the density is highest, to the center, and the rest of the points on either side of the center?

I thought about it, but soon realized how absurd that idea was. I am asking for a transformation which would throw points in the middle and on the right in [0,1] to either side of the mean (=0 for N(0,1)). I am essentially asking for a non-monotonic transformation, which cannot be good, because then there is little meaning left to your transformed feature values. Sure, you might get a bell shaped distribution, but there is no meaning to the transformed values, since the ordering is no longer preserved (see the scatter plot for the transformed feature values in Figure 3). There is no meaning for a sample’s feature value to be higher than the other’s, because after the transform, that ordering may or may not be preserved.

```
log_transform = lambda ar: np.multiply(1.6 * np.log10(ar+1e-8),
np.random.choice((-1, 1), size=ar.size))fontdict = {'family':'serif',
'color':'darkgreen', 'size':16}fig, axs = plt.subplots(1, 1, figsize=(8,
8))axs.hist(standard_normal, bins=1_000, density=True, fc=(0,0,1,0.4),
label='Standard Normal')axs.hist(log_transform(sample), bins=1_000, density=True,
fc=(1,0,0,0.4), label='Log Transform')axs.scatter(log_transform(x),
np.full_like(x, 3e-3), c=x, cmap=cmap)axs.set_xlim(-5, 5)axs.set_title('Log
Transform', fontdict=fontdict, fontweight='bold',
pad=12)axs.set_xlabel('$\pm$1.6log(X)', fontdict=fontdict, fontweight='normal',
labelpad=12)*axs.set_ylabel('Density', fontdict=fontdict, fontweight='normal',
labelpad=12)axs.legend()axs.grid()
```

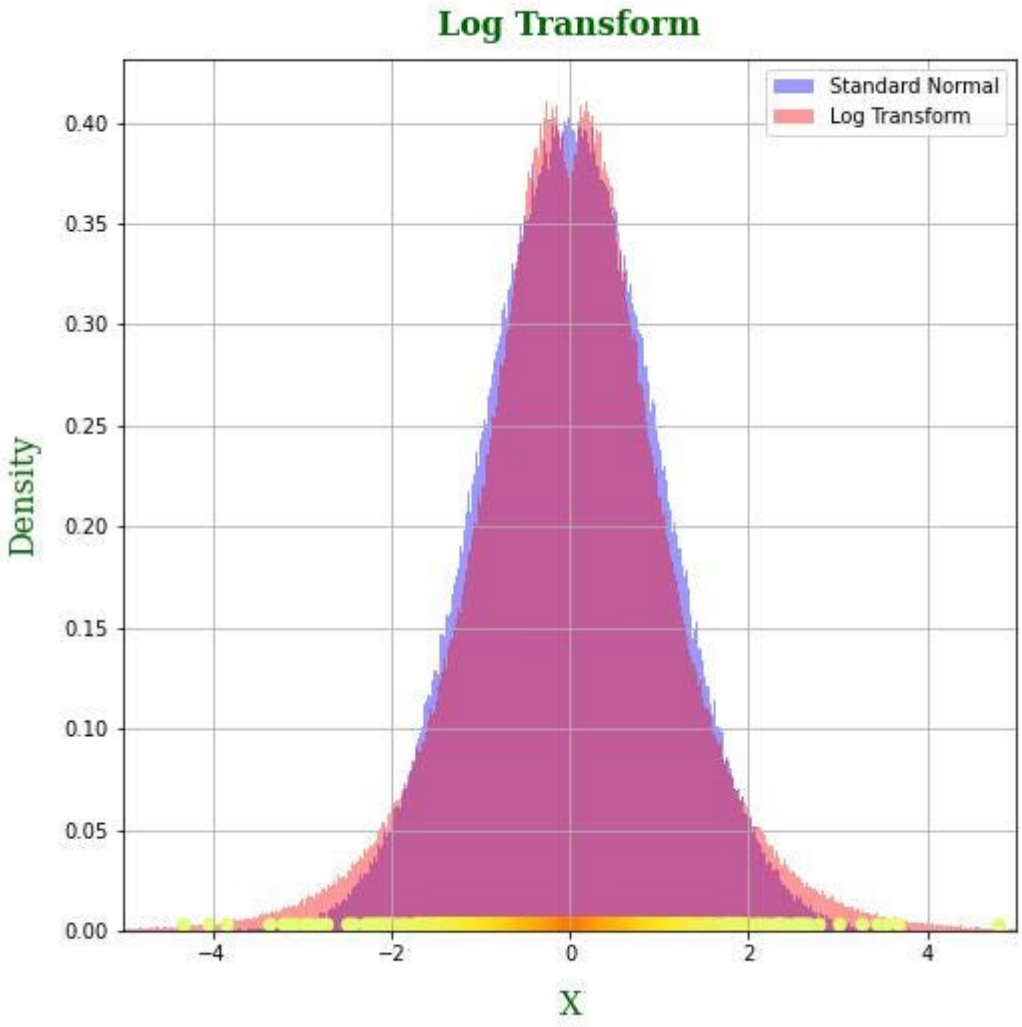

Figure 3: Density plot for the log transformed feature

I thought more about it. It bugged me to accept that is not possible. Why not? My feature has a monotonically decreasing density. My goal is to stretch and squish the [0,1] range around different points using a transform with range (-∞, ∞), and the density at each point in the transformed space should be the as given by N(0,1). This sparked an idea in my mind!

Using percentiles

If I relax my goal, and just try to make sure that the transform function maps the points between the (i-1)th and ith percentile of the original distribution to somewhere between the (i-1)th and ith percentile of N(0,1), that'd be progress.

$$F(x) = \frac{3x}{2} - \frac{x^2}{2};\ x \in [0,1]$$

Equation 1: CDF of the original feature

$$g: [0,1] \to \mathbb{R}\ \text{st.}\ \forall i \in \{1,2,\dots,100\}, g^{-1}\left(\left(\Phi^{-1}\left(\frac{i-1}{100}\right), \Phi^{-1}\left(\frac{i}{100}\right)\right)\right) = \left[F^{-1}\left(\frac{i-1}{100}\right), F^{-1}\left(\frac{i}{100}\right)\right]$$

Equation 2: g is the transform we are looking for and Φ is the CDF of N(0,1)

But this made me realize, my final goal is just an extension of this method. Instead of limiting my method to intervals defined by percentiles, I want a function which would satisfy this for each interval as in Equation 3.

$$g: [0,1] \to \mathbb{R}\ \text{st.}\ \forall x, y \in [0,1]\ \text{with}\ x < y, g^{-1}\left(\left(\phi^{-1}(x), \phi^{-1}(y)\right)\right) = [F^{-1}(x), F^{-1}(y)]$$

Equation 3: g is the transform we are looking for and Φ is the CDF of N(0,1)

Using Uniqueness of Distribution Function

For those with a background in probability theory, recall that a probability is characterized by its distribution function (Theorem 7.1 from Probability Essentials by Jean Jacod and Philip Protter).

I will restrict myself to the space of monotonically increasing functions.

$$H = \{g: [0,1] \to \mathbb{R} \mid \forall x, y \in [0,1]\ \text{with}\ x < y, g(x) \leq g(y)\}$$

Equation 4: Constrained hypothesis set of monotonically increasing functions

If I can find a function such that for the CDF of the transformed feature is equal to the CDF of N(0,1), then I am done. This, along with the monotonically increasing constraint in Equation 4, gives us Equation 5below.

$$F(x) = \Phi\big(g(x)\big);\ x \in [0,1]$$
$$\Rightarrow g = \phi^{-1} \circ F$$

Equation 5: Transform function, g, as a composition of inverse of Φ and F

Results

I used the result in equation 4 to transform my feature so that it has a Standard Normal distribution.

```
fontdict = {'family':'serif', 'color':'darkgreen', 'size':16}fig, axs =
plt.subplots(1, 1, figsize=(8, 8))axs.hist(standard_normal, bins=1_000,
density=True, fc=(0,0,1,0.4), label='Standard
Normal')axs.hist(scipy.stats.norm.ppf(1.5*sample - 0.5*(sample**2)), bins=1000,
density=True, fc=(1,0,0,0.4), label='Equation 4
Transform')axs.scatter(norm.ppf(1.5*x - 0.5*(x**2)), np.full_like(x, 3e-3), c=x,
cmap=cmap)axs.set_xlim(-5, 5)axs.set_title("Transformed Feature's Density",
fontdict=fontdict, fontweight='bold', pad=12)axs.set_xlabel('$\Phi^{-1}(F$(X))',
fontdict=fontdict, fontweight='normal', labelpad=12)axs.set_ylabel('Density',
fontdict=fontdict, fontweight='normal', labelpad=12)axs.legend()axs.grid()
```

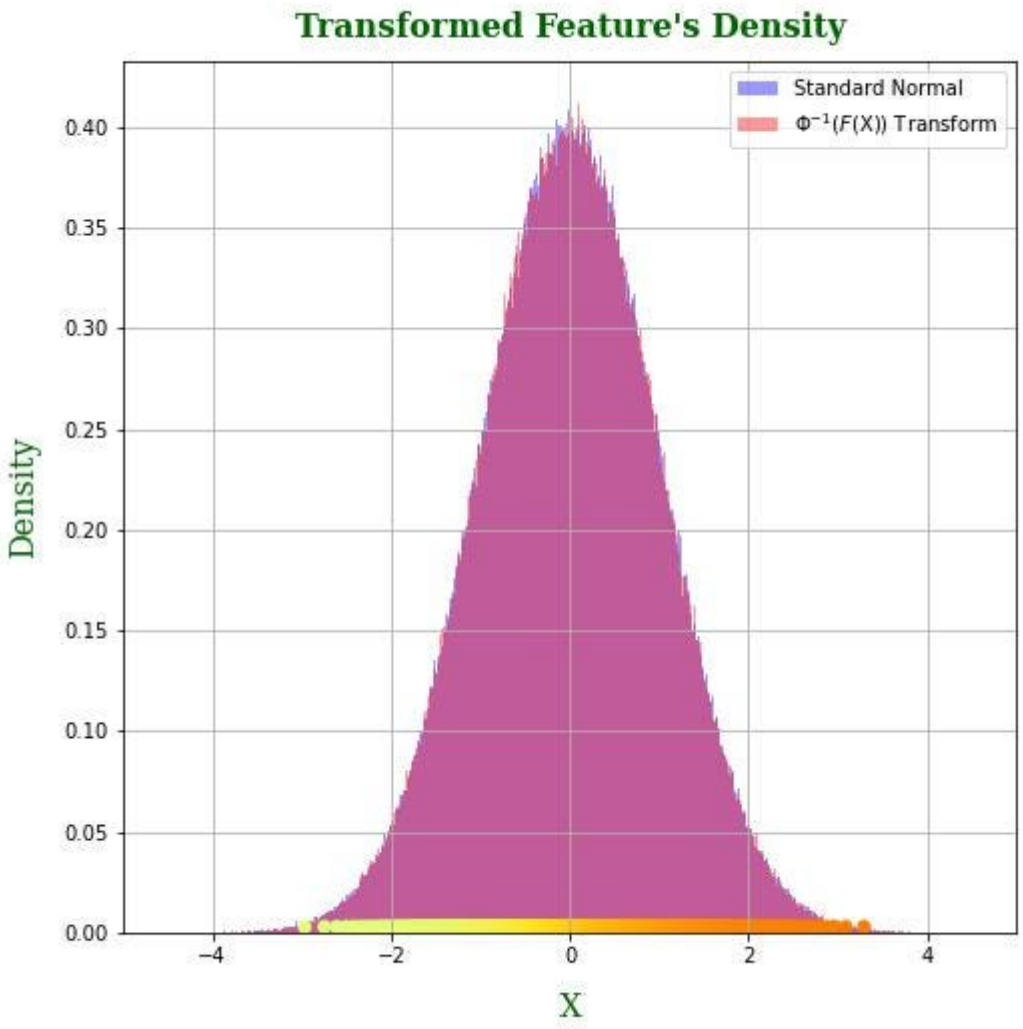

Figure 4: Density of the transformed feature using Equation 4

Word of Caution

You can do this with any distribution as long as it has a continuous distribution function. But before using this, please check the shape of the transform and see if it makes sense to use it for your use-case. Don’t simply use it because it gives you a Gaussian distribution.

```
fontdict = {'family':'serif', 'color':'darkgreen', 'size':16}fig, axs =
plt.subplots(1, 1, figsize=(8, 8))axs.scatter(x, norm.ppf(1.5*x - 0.5*(x**2)),
c=x, cmap=cmap)axs.set_xlim(0, 1)axs.set_title('Transform', fontdict=fontdict,
fontweight='bold', pad=12)axs.set_xlabel('X', fontdict=fontdict,
fontweight='normal', labelpad=12)axs.set_ylabel('$\Phi^{-1}(F$(X))',
fontdict=fontdict, fontweight='normal',
labelpad=12)axs.grid()plt.savefig('final_transform.jpg')
```

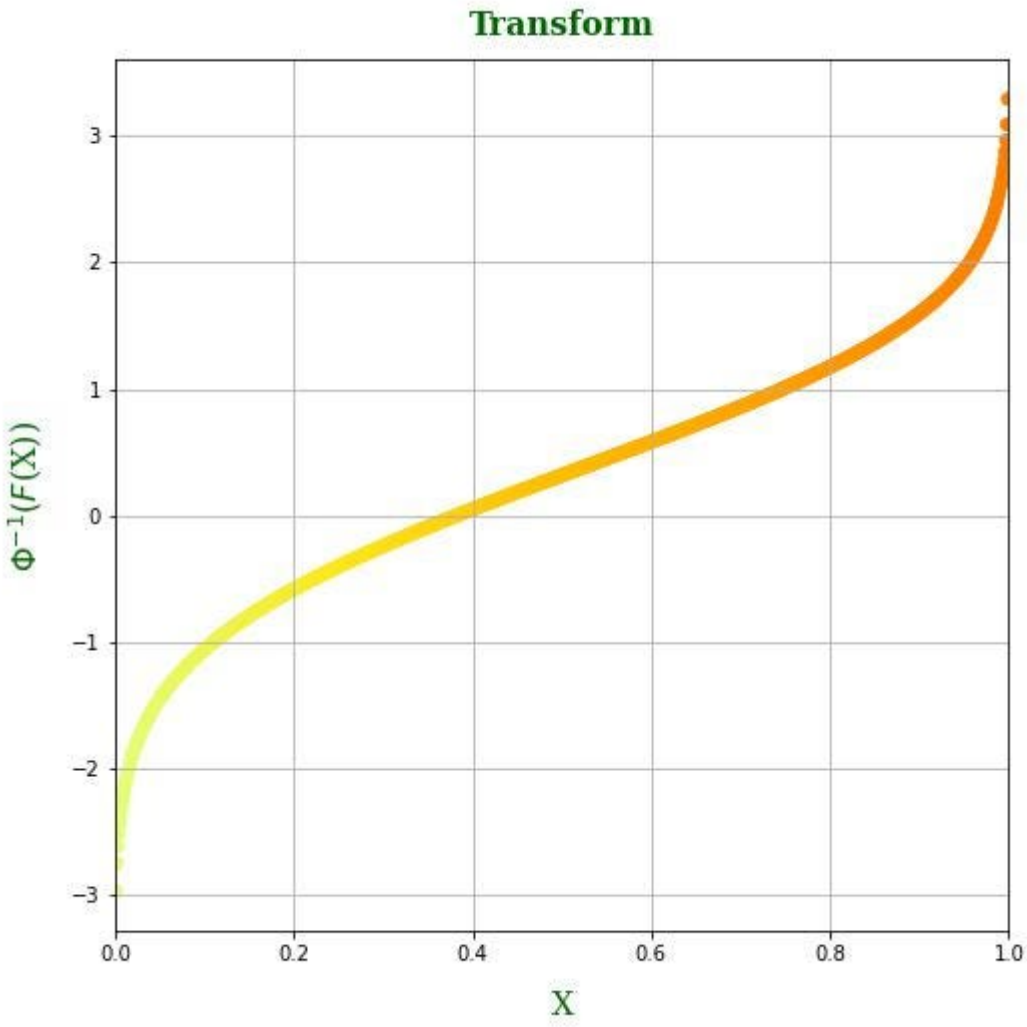

Figure 5: Transform Function

For instance, this process gave me a transform as in Figure 5. I'd only use this transform if the output is much more volatile when this feature takes values near 0 or near 1, but less volatile when the values are near 0.5. If this is not the case, I would be feeding the model an incorrect interpretation of the feature, likely hurting its performance.

Hope this helps!

PS. Kindly let me know if you find any mistakes (grammatical, factual, gaps in explanation, etc.) or if you have suggestions for me to improve this article (elaborating any idea, adding new ideas, etc.). I'd be happy to discuss and update accordingly! :)