

# **IIT Madras**

## **ONLINE DEGREE**

**Machine Learning Practice**  
**Online Degree Programme**  
**B. Sc in Programming and Data Science**  
**Diploma Level**  
**Dr. Ashish Tendulkar**  
**Indian Institute of Technology – Madras**

**Numeric Transformers**

**(Refer Slide Time: 00:12)**





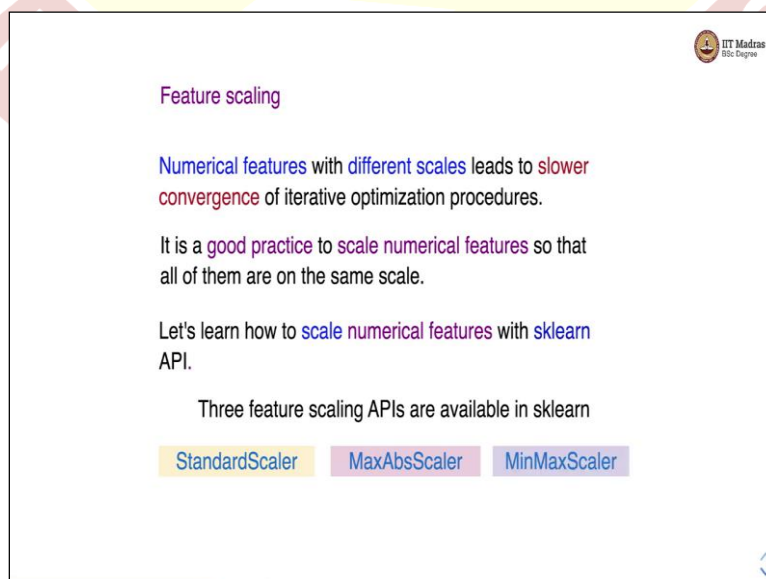
**1.2 Numeric transformers**


- 1. Feature scaling
- 2. Polynomial transformation
- 3. Discretization



Namaste welcome to the next video of the machine learning practice course. In this video, we will discuss numeric transformers in data pre-processing. Numeric transformers are applied to numerical features. There are three types of transformers the first one is feature scaling second is polynomial transformers and the third one is discretization transformers.

**(Refer Slide Time: 00:40)**





**Feature scaling**


Numerical features with different scales leads to slower convergence of iterative optimization procedures.

It is a good practice to scale numerical features so that all of them are on the same scale.

Let's learn how to scale numerical features with sklearn API.

Three feature scaling APIs are available in sklearn

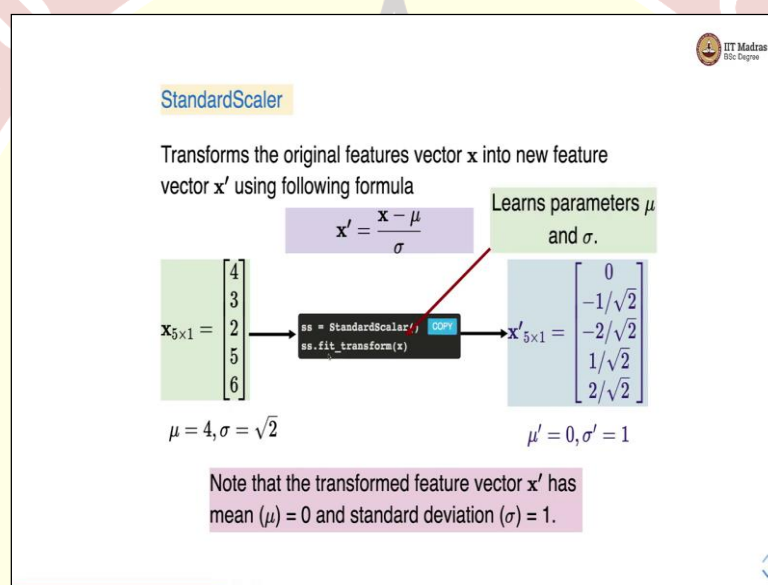
- StandardScaler
- MaxAbsScaler
- MinMaxScaler



Let us look at feature scaling. Numerical features with different scales lead to slower convergence of iterative optimization procedures. Hence it is a good practice to scale numerical features so that all of them are on the same scale. This leads to better convergence of iterative optimization procedures like gradient descent, mini-batch gradient descent, and stochastic gradient descent.

Let us learn how to scale numerical features with sklearn API. There are three feature scaling APIs that are available in sklearn. Standard scalar, max absolute scalar, and min-max scalar.

**(Refer Slide Time: 01:31)**



Let us look at them one by one. The standard scalar transforms the original feature vector into a new feature vector by using the following formula. We calculate mu which is the mean and the standard deviation and basically subtract the mean from the future value and divide the difference by the standard deviation to get the transformed value. Let us take an example.

We have an original feature matrix that has got 5 samples and a single feature. We can calculate the mean and standard deviation from this particular feature column. The mean is 4 and the standard deviation is a square root of 2. We first instantiate the standard scalar object and call the fit\_transfer method on it with the original feature matrix as an argument. Again I would request you to notice this fit\_transform and also appreciate the uniformity of the Sklearn API.

For all transformers, we are using the fit\_transfer method on the original feature matrix. And this gives us basically the transform feature matrix where each value is transformed according to this particular formula. So, you can see that in the case of the first value the value is 4 the mean is also 4. So,  $4 - 4$  is 0. So, we get a value of 0. In the case of the second value, this difference is  $3 - 4$  which is -1 and we divided by the standard deviation which is the square root of 2.

So, it is -1 by the square root of 2 and so on. So, you can also see that the remaining values also come from this formula you can verify that by doing the computation yourself. And note that in the case of the transform feature vector we have a mean 0 and standard deviation of 1. So, we convert the feature column with a mean of 4 and standard deviation of the square root of 2 to mean 0 and standard deviation 1.

So, this is what standard scalar does to each feature column or to each feature. And recall that this fit\_transfer method learns parameter mu and standard deviation from the original feature column that is specifically done by the fit method. And those parameters are actually stored in order to they are stored in this particular object of standard scalar and these parameters are used again and again in order to transform this particular feature column in other sets like eval set and the test set.

**(Refer Slide Time: 04:55)**

### MinMaxScaler

It transforms the original feature vector  $x$  into new feature vector  $x'$  so that all values fall within range  $[0, 1]$

$$x' = \frac{x - x.min}{x.max - x.min}$$

where  $x.max$  and  $x.min$  are largest and smallest values of that feature respectively, of the original feature vector  $x$ .

$x_{5 \times 1} = \begin{bmatrix} 15 \\ 2 \\ 5 \\ -2 \\ -5 \end{bmatrix}$

$\xrightarrow{\begin{matrix} \text{rms} = \text{MinMaxScaler}() \\ \text{rms.fit\_transform}(x) \end{matrix}}$

$x'_{5 \times 1} = \begin{bmatrix} 1 \\ 0.35 \\ 0.5 \\ 0.6 \\ 0 \end{bmatrix}$

$x.max = 15, x.min = -5$

The largest number is transformed to 1 and the smallest number is transformed to 0.

The min max scalar is another way to perform feature scaling. It transforms the original feature vector into a new feature vector. So, that all values fall within range 0 to 1, so, it achieves it with the following formula. It subtracts the value of each feature with it subtract

the value of each sample with the minimum value of the feature and divides it by the range the total range for that feature that is calculated by subtracting the min value from the max value.


So,  $x_{\max}$  and  $x_{\min}$  are the largest and smallest value of that feature of course this is in the original feature vector. Let us take a concrete example this is the original feature matrix containing five samples and of course, there is a single feature. So, we first before applying the min-max scalar we first instantiate the min-max scalar object and call the `fit_transfer` method on the original feature set.

And this gives us the transform feature matrix with the transformed values that are computed using this particular formula where each feature value is; so, from each feature value, we subtract the minimum value and divide the difference by the range of the feature. Here you can see that the range of the feature is from -5 to 5. So,  $x_{\min}$  is -5, and  $x_{\max}$  is 5. So, the total range is 10 and you can see that the maximum value is 5.

So, the maximum value gets transformed to 1 because this is  $5 - (-5)$ . So, this is the numerator becomes 10 and the range is 10. So, it is 10 by 10 which is one and the minimum value basically gets transformed to zero and you can again verify that using this formula. So, the value is -5 we subtract  $-5 - (-5)$  this is 0. So, hence it gets transformed to 0. You can also check this let us say for this value 5.

So, we have a value of 5 and the minimum is -5. So, the numerator is 10 and the range is 10. So, it is 10 by 10 hence this 5 gets transformed to 1.

**(Refer Slide Time: 07:43)**

 IIT Madras  
BSc Degree

### MaxAbsScaler

It transforms the original features vector  $x$  into new feature vector  $x'$  so that all values fall within range  $[-1, 1]$

$$x' = \frac{x}{\text{MaxAbsoluteValue}}$$

where  $\text{MaxAbsoluteValue} = \max(x.\text{max}, |x.\text{min}|)$

$x_{5 \times 1} = \begin{bmatrix} 4 \\ 2 \\ 5 \\ -2 \\ -100 \end{bmatrix}$

$\rightarrow$

```
mas = MaxAbsScaler()
mas.fit_transform(x)
```

$\rightarrow$

$x'_{5 \times 1} = \begin{bmatrix} 0.04 \\ 0.02 \\ 0.05 \\ -0.02 \\ -1 \end{bmatrix}$

$\text{MaxAbsoluteValue} = \max(5, |-100|) = 100$


Max absolute scalar is the third feature scalar and it transforms the original feature vector into a new vector. So, that all values fall within the range -1 to +1. So, here we divide each value of the feature by the maximum absolute value and the maximum absolute value is calculated as max of x.max and mode of x.mean. So, x.max and we also take the mode of x.mean and we calculate what is the max value.

So, let us take a concrete example here we have a feature matrix with 5 samples and one feature these are the feature values you can see that the maximum absolute value here is the max of the maximum value is 5 and the minimum value is -100. So, we take the mode of -100 which is 100. So, the maximum absolute value is 100 and when we apply fit transform on the max absolute scalar object we get a transformed feature matrix which is with these values.

So, you can see that the maximum absolute value is 100 and hence this particular value gets transformed to -1. So, the only difference between the previous scalar and this scalar is that this scalar gets us values in the range -1 to +1 earlier scalar was getting us a value between 0 to 1.

**(Refer Slide Time: 09:29)**





**FunctionTransformer**

Constructs transformed features by applying a user defined function.

$X_{4 \times 2} = \begin{bmatrix} 128 & 2 \\ 2 & 256 \\ 4 & 1 \\ 512 & 64 \end{bmatrix}$

$X'_{4 \times 2} = \begin{bmatrix} 7 & 1 \\ 1 & 8 \\ 2 & 0 \\ 9 & 6 \end{bmatrix}$

```
ft = FunctionTransformer(numpy.log2)
ft.fit_transform(X)
```

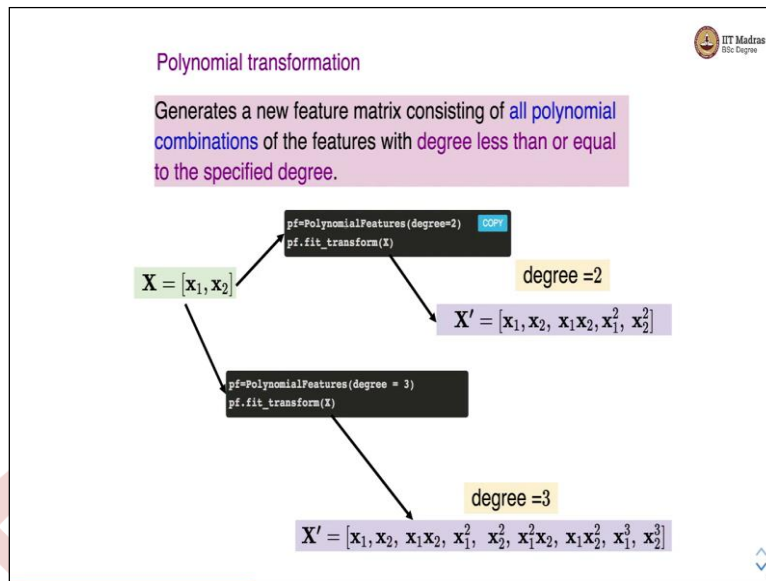
Applies  $\log_2$  function to the features.

So these are some specific features scaling techniques that we are using we can also transform the feature using certain transformers like log transform or square root transform or any other custom transformer of your choice. So, here the function transformer helps us to construct the transform feature by applying a user-defined function. Let us take a concrete example here we have an original feature matrix with four samples and two features.

So, this is the first feature the first column represents the first feature second column represents the second feature. Here we instantiate a function transformer object with a function that is the  $\log_2$  and then we call the `fit_transform` on this original feature matrix. So, what happens is we apply the log to the base 2 to each feature and obtain the transform feature set.

So, you can see here that this 128 is transformed to 7 because when  $\log_2(128)$  is 7 then the  $\log_2(2)$  is 1  $\log_2(4)$  is 2  $\log_2(512)$  is 9, and so on. So, you can also apply this log to the base 2 to each value in the second feature column and you will see the transformed values which are log to the base 2 of the original values.

**(Refer Slide Time: 11:20)**



We have seen in the machine learning techniques course why polynomial transformations are important to us. Polynomial transformations help us to learn the non-linear relationship between the features and the label. So, we use polynomial transformation to generate a new feature matrix consisting of all polynomial combinations of the features with degrees less than or equal to the specified degree.

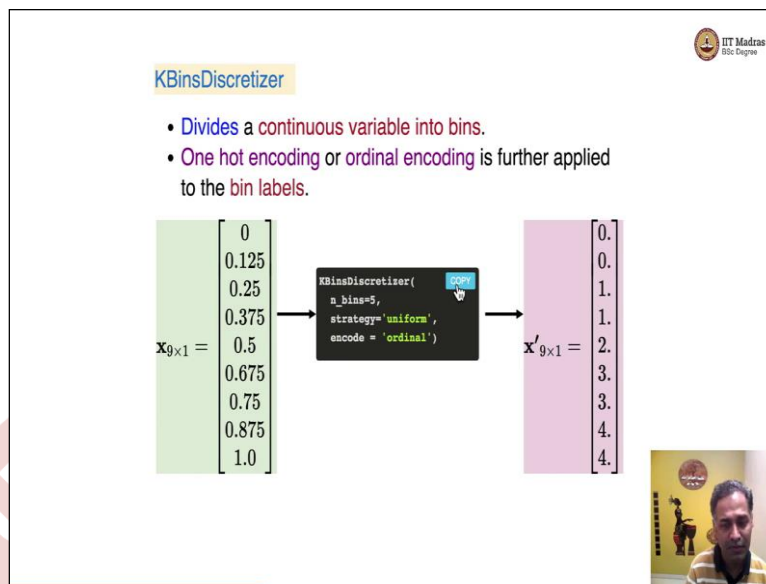
Let us take a concrete example. Let us say we have a feature matrix with two features  $x_1$  and  $x_2$ . Let us apply polynomial features of degree 2. So, we instantiate a polynomial features object with degree 2 and we apply `fit_transfer` method on this original feature matrix. So, what happens is we get the degree two transformations here which are  $x_1$  is copied over as it is  $x_2$  is copied over as it is then you have  $x_1, x_2, x_1^2, x_2^2$ .

In addition to that, we also get a value of 1 here which is not shown in this particular representation. In the same way, we can apply polynomial features of degree 3 and when we apply polynomial features of degree 3 you can see that we have now the degree 3 feature the maximum degree of the polynomial is 3. So, you can see that the features here are  $x_1$  and  $x_2$  are copied over as it is there will also be 1 which is copied over by default.

So, there is  $1 \times 1 \times 2$  then the combination  $x_1, x_2, x_1^2, x_2^2$  then we have the third-order features  $x_1^2 \times x_2, x_1 \times x_2^2, x_1^3$  and  $x_2^3$ . So, you can also see that degree 3 polynomials has all the features from degree 2 polynomials and additional features from the third degree. So, we have this polynomial features transformer that can be used to convert the original feature matrix into a polynomial of the desired degree.



(Refer Slide Time: 13:42)



Then there is KBinsDiscretizer that divides a continuous variable into bins and each bin is represented with one-hot encoding or with ordinal encoding. Let us look at a concrete example. Let us say we have a feature matrix with 9 samples and 1 feature and these are values for that feature for each of the samples. Let us say we decide to use KBinsDiscretizer with the number of bins equal to 5.

We want to split these bins uniformly and we want to use Let us say ordinal encoding. So, what happens is the entire range is split into five different parts and based on that the transform features are calculated. So, you can see that since the range is between 0 to 1 there is we can make 5 bins with you know with thresholds of 0.2, 0.4, 0.6, 0.8, and 1 and you can see that values that are less than 0.2 are transformed to value 0.

So, these first two values are transformed to value 0 then we have values between 0.2 to 0.4 transform to 1, 0.4 to 0.6 there is only one value which is transformed to 2, 0.6 to 0.8 is transformed to 3 and 0.8 to 1 is transformed to 4. So, this is how KBinDiscretizer works. Sometimes we have to need to convert the numerical features into such kind of discrete features and KBinsDiscretizer will help us perform that transformation.