

IIT Madras

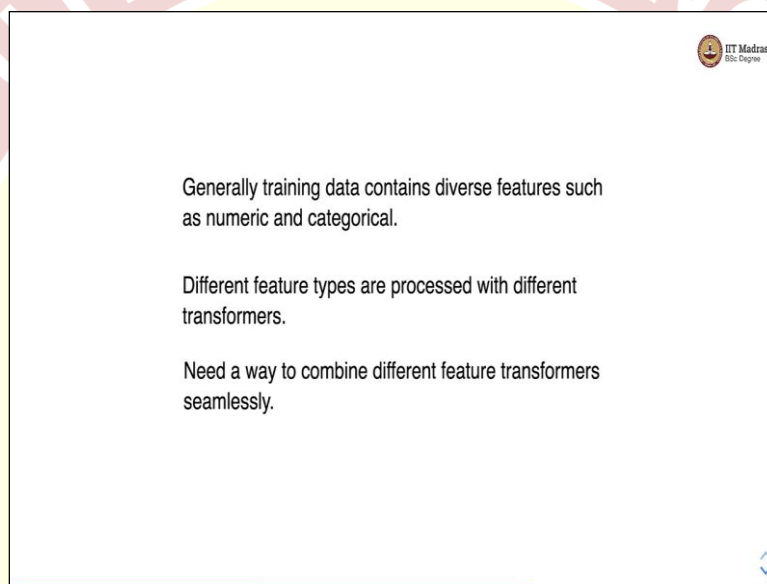
ONLINE DEGREE

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

Heterogeneous Features Transformations


Namaste welcome to the next video of the machine learning practice course. In this video, we will discuss how to apply transformations to diverse feature sets.

(Refer Slide Time: 00:22)



So, generally, training data contains diverse features such as numerical and categorical features. Different types of features require different transformations and we need a way to seamlessly combine these different transformers.


(Refer Slide Time: 00:38)



Composite Transformer


`sklearn.compose` has useful classes and methods to apply transformation on subset of features and combine them:

`ColumnTransformer` `TransformedTargetRegressor`




So, sklearn provides composite transformer `sklearn.compose` module has useful classes and methods that we can apply to a subset of features and combine them two classes will focus on one is `ColumnTransformer` and the second is `TransformTargetRegressor`.

(Refer Slide Time: 01:01)



ColumnTransformer

- It applies a set of transformers to columns of an array or `pandas.DataFrame`, concatenates the transformed outputs from different transformers into a single matrix.
- It is useful for transforming heterogenous data by applying different transformers to separate subsets of features.
- It combines different feature selection mechanisms and transformation into a single transformer object.



Let us first look at `ColumnTransformer`. `ColumnTransformer` applies a set of transformers to columns of an array or Pandas data frame. It concatenates a transformed output from different transformers into a single matrix. The `ColumnTransformers` are useful for transforming heterogeneous data by applying different transformers to separate subsets of features.

It combines different feature selection mechanisms and transformations into a single transformer object.

(Refer Slide Time: 01:38)



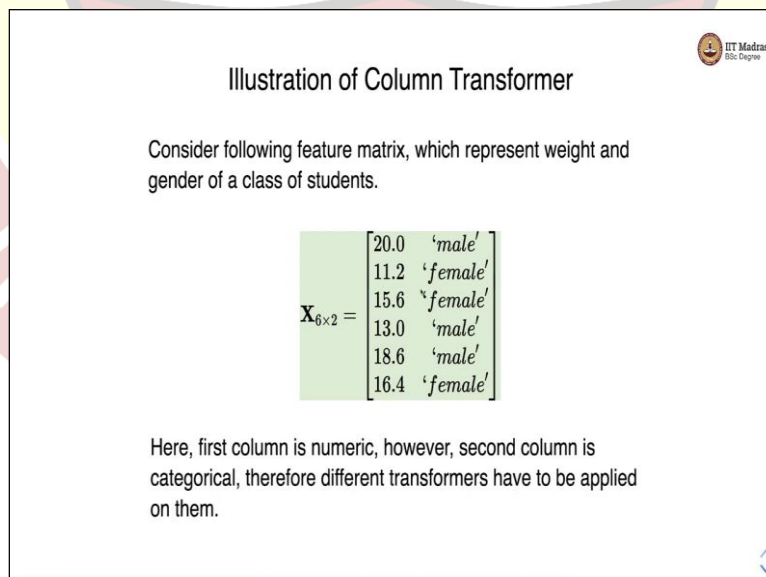
IT Madras
BSc Degree

- `ColumnTransformer()`
- Each tuple has format
 - `(estimatorName, estimator(...), columnIndices)`

```
column_trans = ColumnTransformer(  
    [ ('ageScaler', CountVectorizer(), [0]) ],  
    [ ('genderEncoder', OneHotEncoder(dtype='int'), [1]) ],  
    remainder='drop', verbose_feature_names_out=False)
```

Let us look at a concrete example of how to specify the ColumnTransformer. So, here we have an example of a ColumnTransformer. We first instantiate an object of ColumnTransformer by specifying different transformations that we want to apply on different columns. So, here is the name of the transformer this is the actual transformer which is instantiated here and this is the index the feature index of the column index on which this transformer has to be applied.

(Refer Slide Time: 02:25)



IT Madras
BSc Degree

Illustration of Column Transformer

Consider following feature matrix, which represent weight and gender of a class of students.

$$\mathbf{X}_{6 \times 2} = \begin{bmatrix} 20.0 & 'male' \\ 11.2 & 'female' \\ 15.6 & 'female' \\ 13.0 & 'male' \\ 18.6 & 'male' \\ 16.4 & 'female' \end{bmatrix}$$

Here, first column is numeric, however, second column is categorical, therefore different transformers have to be applied on them.

Consider the following feature matrix as a concrete example. We have six samples and two features. The first feature represents the weight and the second feature represents the gender of students in the class the first column is numeric and the second column is categorical. So, we need different transformers for each of these features.

(Refer Slide Time: 02:55)

In this example, let's apply `MaxAbsScaler` on the numeric column and `OneHotEncoder` on the categorical column.

```
column_trans = ColumnTransformer(  
    [('ageScaler', MaxAbsScaler(), [0]),  
     ('genderEncoder', OneHotEncoder(dtype='int'), [1])],  
    remainder='drop', verbose_feature_names_out=False)  
column_trans.fit_transform(X)
```

$X_{6 \times 2} = \begin{bmatrix} 20.0 & \text{'male'} \\ 11.2 & \text{'female'} \\ 15.6 & \text{'female'} \\ 13.0 & \text{'male'} \\ 18.6 & \text{'male'} \\ 16.4 & \text{'female'} \end{bmatrix}$

$X'_{6 \times 3} = \begin{bmatrix} 1. & 0. & 1. \\ 0.56 & 1. & 0. \\ 0.78 & 1. & 0. \\ 0.65 & 0. & 1. \\ 0.93 & 0. & 1. \\ 0.82 & 1. & 0. \end{bmatrix}$

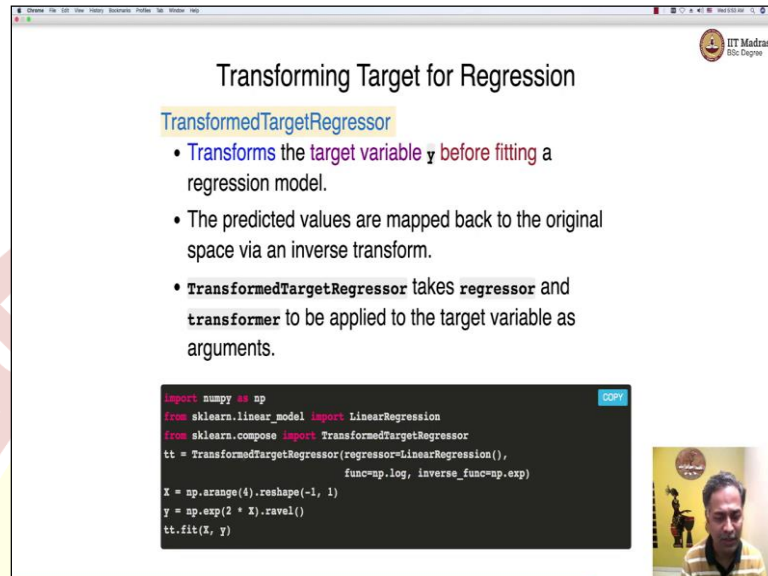
In this example let us apply `MaxAbsoluteScalar` on the numerical column and `OneHotEncoder` on the categorical column this is our original feature matrix we apply `Max Absolute Scalar` on the first column and the `OneHotEncoder` on the second column. So, we specify the max absolute scalar on the first column and `OneHotEncoder` along with its argument on the second column we give the names to each of these transformers the first transformer is named as a scalar and the second one is named as gender encoder.

So, we can use these names to refer to these transformers in the `ColumnTransformer` object. So, after applying column transformation you can see that the first column is transformed according to `Max Absolute Scalar` and is the maximum value which got transformed to the value of 1 and other values are scaled according to the Maximum Absolute Value whereas the second column which was a categorical column is converted into one-hot encoding representation.

So, since there are two unique values male and female. So, this single-column is transformed into two columns. The first column corresponds to female and the second column corresponds to male you can see that wherever there is a male the second column or in this transform feature representation the third column is one wherever there is male in that corresponding sample.

And whenever there is a there is female the second column of the transform feature matrix is one. So this is how the ColumnTransformer works it allows us to apply different transformations to different feature columns based on their types.

(Refer Slide Time: 05:00)



Transforming Target for Regression

TransformedTargetRegressor

- Transforms the target variable y before fitting a regression model.
- The predicted values are mapped back to the original space via an inverse transform.
- TransformedTargetRegressor takes regressor and transformer to be applied to the target variable as arguments.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.compose import TransformedTargetRegressor
tt = TransformedTargetRegressor(regressor=LinearRegression(),
                                func=np.log, inverse_func=np.exp)
X = np.arange(4).reshape(-1, 1)
y = np.exp(2 * X).ravel()
tt.fit(X, y)
```

The second class is Transformedtargetregressor. It transforms a target variable before fitting a regression model. The target variable is also referred to as a label. The predicted values are mapped back to the original space via inverse transform this is very important because we will be making predictions in the transform space but we want those predictions translated back into the original space and that is done to the inverse transform.

So, Transformedtargetregressor takes regressor and transformer as inputs. So, let us take a concrete example. So, here we instantiate an object of transform target regressor by specifying a linear regression as a regressor and we specify the function the transformation function to be np.log. So, we want to apply a log transformation on the label.

And we also specify what is the inverse transform. So, exponential is the inverse of log hence we specify exponential as the inverse transform. And we generate some data randomly and labels and we call the fit method on this transform target regressor by providing the feature matrix and the label vector. So, this tt.fit first applies a transformation on the target, and then it performs the fit of the feature matrix and the transform target.

This is how to transform target regressor works and whenever we predict with transform target regressor we will get output in the transform space which we translate back into the original space using the inverse function that is specified as an argument of the Transformedtargetregressor.

