

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

Categorical Transformers

(Refer Slide Time: 00:12)

IIT Madras
BSc Degree

1.2 Categorical transformers

1. Feature encoding
2. Label encoding

IIT Madras
BSc Degree

Namaste welcome to the next video of the machine learning practice course. In this video, we will discuss categorical transformers. We will use categorical transformers for categorical feature encoding and also for label encoding.

(Refer Slide Time: 00:28)

IIT Madras
BSc Degree

OneHotEncoder

- Encodes **categorical feature** or **label** as a **one-hot** numeric array.
- Creates **one binary column** for each of K unique values.
- **Exactly one column** has 1 in it and rest have 0.

$x_{4 \times 1} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$

$\xrightarrow{\begin{matrix} ohe = \text{OneHotEncoder}() \\ ohe.fit_transform(x) \end{matrix}}$

$x'_{4 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

unique values:

$K = 3$

columns in
transformed matrix = 3

IIT Madras
BSc Degree

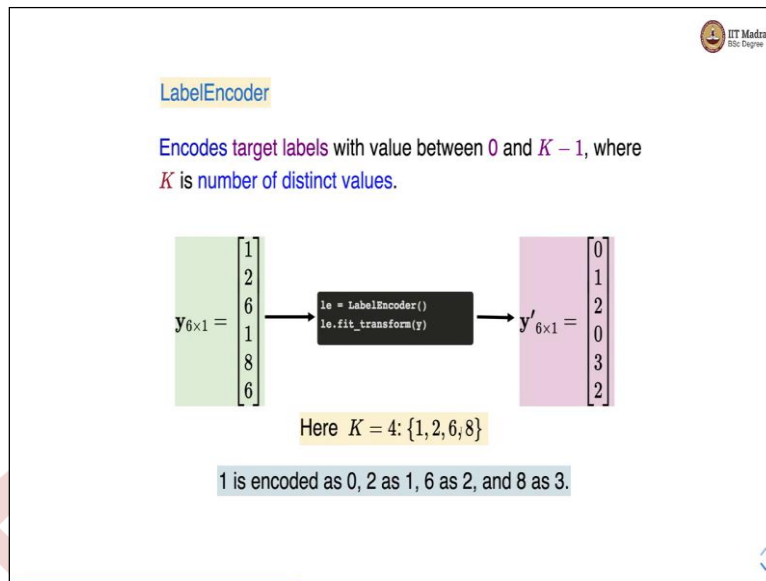
The first categorical transformer is OneHotEncoder. OneHotEncoder encodes categorical features or labels as a 1 hot numeric array. It creates 1 binary column for each of K unique values out of these K binary columns only 1 column has 1 in it and the rest (K - 1) columns have zeros in it. Let us look at a concrete example. Let us say we have a feature matrix which is (4×1) .

There are 4 samples and it is a single feature this particular feature has got 4 values 1, 2, 3, and 1. The number of unique values here is equal to 3 number of unique values which is represented by K is equal to 3. Hence in the transform matrix, we will create 3 columns for this particular single column each column corresponds to a unique value. We first instantiate the OneHotEncoder object and call the fit_transform method on the original feature matrix again remember this is the fit_transform method because 1 hot encoder is a transformer.

So, we get a transform feature matrix which is 4 cross 3. There are 4 samples and 3 columns in the transform feature matrix and you can see that the first column corresponds to value 1 first unique value which is value 1. The second column corresponds to value 2 and the third column corresponds to value 3.

You can see that the first sample and the last sample have 1 in them hence you see 1 in the first sample and 1 in the last sample in the first column. You can see that 2 appears for the second column for the second sample hence the column corresponding to 2 is 1 and the rest other columns are 0. Here for the third sample, the value is 3 and you will see 1 in the third column which corresponds to the value 3.

(Refer Slide Time: 03:18)

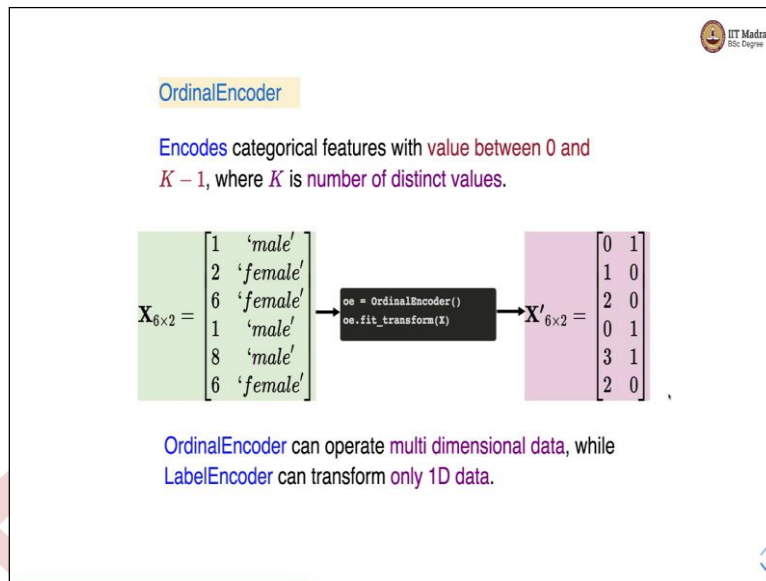


LabelEncoder encodes target variable or target label with a value between 0 and (K-1) where K is the number of distinct values. So, imagine we have a label vector y with 6 samples and it is a vector. So, hence the second dimension is 1 and the values in the label vector are 1, 2, 6, 1, 8, and 6. Here the unique values are 4 unique values are 1, 2, 6, and 8 and hence K is equal to 4.

Now if you use LabelEncoder on this we create an object of LabelEncoder transformer and we call fit_transform method with this label vector as an argument and we get a transformed label vector with 6 samples and it is again a vector hence the second dimension is 1. one here 1 is encoded as zero, 2 is encoded as 1, 6 is encoded as 2 and 8 is encoded as 3 and you can see that right.

So, wherever there was 1 in the original feature matrix we are getting 0 in the transform feature matrix. So, there is 1 for sample 1 and sample 4. So, now we have in the transform feature matrix the value of the label is 0. There were 2 in the second sample which is mapped to 1 then there is 6 in the third sample which is mapped to 2 and 6 is also there in the last sample which is mapped to 2 and 8 is there in the second last sample and that is mapped to 3. So, this is how LabelEncoder works it encodes the target variable with values between 0 and K-1.

(Refer Slide Time: 05:22)



Let us look at the third encoder which is OrdinalEncoder. OrdinalEncoder encodes categorical features with values between 0 and $K - 1$ where K is the number of distinct values. Let us take a concrete example with the original feature matrix with 6 samples and 2 features. The first feature seemed to be a numeric feature and the second feature is categorical. In the first feature, we have values 1, 2, 6, 1, 8, and 6.


And the second column on the second feature has the gender which is male, female, female, male, male, and female. So, we can see that there are 1, 2, 3, 4 unique values for the first feature and 2 unique values for the second feature. We instantiate the OrdinalEncoder object and call fit_transform with the original feature matrix as an argument.

And we obtain a transform feature matrix where each unique value is mapped to its kind of transformed to a value between 0 to $K-1$. So, here 1 is transformed to 0 you can see that wherever there is 1 you can see the value 0 in the transform feature matrix 2 is transformed to 1, 6 is transformed to 2. So, wherever there is 6 in the original feature matrix here and here they are transformed to 2 and 2 over here and 8 is transformed to 3.

Now the second column made is transformed 1 and females are transformed to 0 you can see that wherever there is a there is gender female you will see value 0. In the transform feature matrix and wherever there is a male you will see value 1 in the transform feature matrix. So, the difference between OrdinalEncoder and the label encoder is that the OrdinalEncoder can operate on multi-dimensional data whereas the label encoder can transform only 1-dimensional data.

So, whenever you have multi-dimensional data and you want to do the encoding you can use OrdinalEncoder.

(Refer Slide Time: 07:59)



LabelBinarizer

Several regression and binary classification can be extended to multi-class setup in **one-vs-all** fashion.

This involves training a single regressor or classifier per class.

For this, we need to **convert multi-class labels to binary labels**, and **LabelBinarizer** performs this task.

$y_{6 \times 1} = \begin{bmatrix} 1 \\ 2 \\ 6 \\ 1 \\ 8 \\ 6 \end{bmatrix}$

```
lb=LabelBinarizer()
lb.fit_transform(y)
```

$Y'_{6 \times 4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

If estimator supports multiclass data, **LabelBinarizer** is not needed.

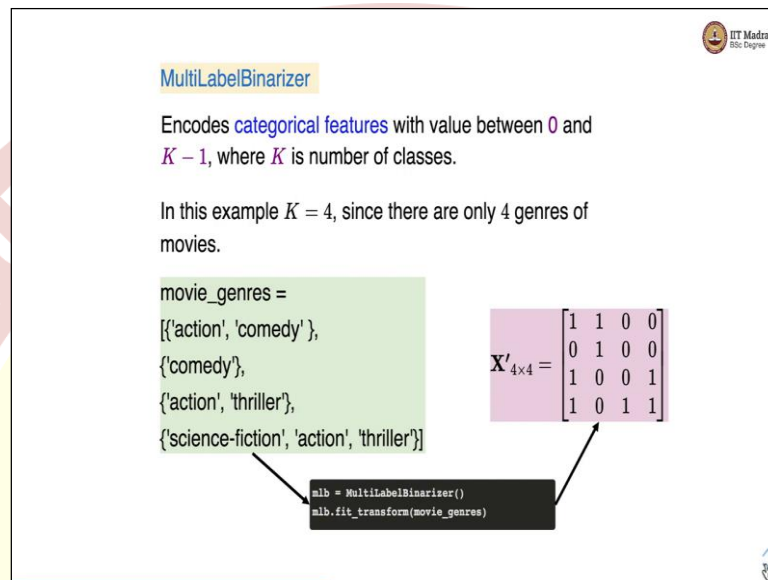
Let us look at the next encoder which is LabelBinarizer several regression and binary classification can be extended to multi-class setup in 1 versus old fashioned and we have seen this in the first week of the machine learning techniques course. Now this involves training a single regressor or classifier per class. So, what happens is that in our data we have multiple classes but we want to train this multi-class setup in one versus old fashion.

In that case, we need to convert multi-class labels to binary labels and sklearn provides this LabelBinarizer class for performing this task. So, let us take a concrete example where we have original feature matrix y this is original feature vector y with 6 samples and this label vector has values 1, 2, 6, 1, 8, and 6. We instantiate the LabelBinarizer object and call the fit_transfer method with this feature with this label vector as an argument and we get a transformed label matrix.

So, this label vector is converted to a label matrix with the same number of samples but the number of columns is now equal to the number of unique values in this particular column. So, you can see that unique values are 1, 2, 6, and 8. So, there is a column corresponding to each of the unique values the first column corresponds to value 1. The second column corresponds to value 2 third column corresponds to value 6 and the 4th column corresponds to value 8.

Now if you have an estimator that already supports multi-class data we need not use LabelBinarizer. LabelBinarizer has to be used if you want to extend regression and binary classification algorithms that do not support multi-class setup. But if you have an estimator that readily supports multi-class setup then LabelBinarizer is not needed.

(Refer Slide Time: 10:33)



MultiLabelBinarizer

Encodes **categorical features** with value between 0 and $K - 1$, where K is number of classes.

In this example $K = 4$, since there are only 4 genres of movies.

```
movie_genres =
[{'action', 'comedy'},
 {'comedy'},
 {'action', 'thriller'},
 {'science-fiction', 'action', 'thriller'}]
```

`mlb = MultiLabelBinarizer()`
`mlb.fit_transform(movie_genres)`

$X'_{4 \times 4} =$

1	1	0	0
0	1	0	0
1	0	0	1
1	0	1	1

Let us extend the LabelBinarizer to MultiLabelBinarizer it encodes categorical features again in the same manner as LabelBinarizer between values 0 and $K - 1$ where K is the number of classes. So, let us say in this let us take a concrete example of the movie genre and there is the total number of genres equal to which are action, comedy, thriller, and science fiction.

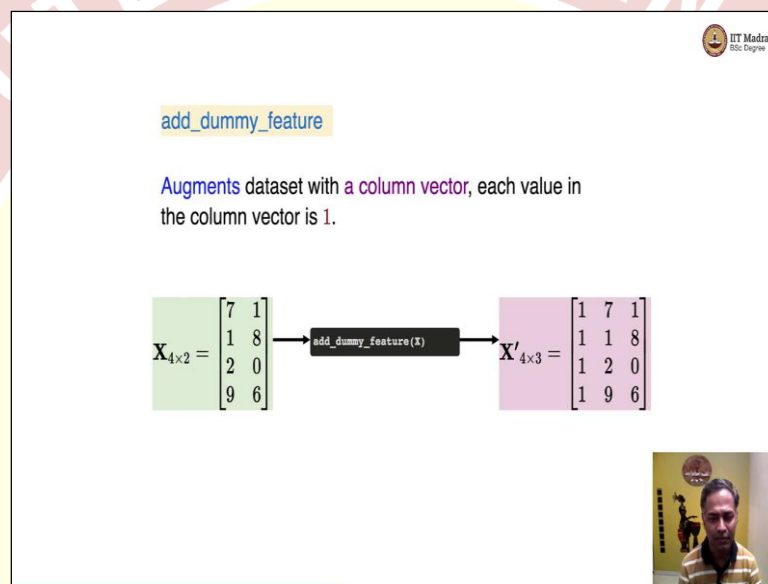
The first sample has got 2 genres which is action and comedy second has a single genre which is comedy third one has action and trailer the 4th one has science fiction action and trailer. Now let us call MultiLevelBinarizer on this particular movie genre and convert them into and encode them with values 0 and $K-1$. We instantiate the object of MultiLevelBinarizer class and call the fit_transfer method with movie genre as an argument and this gets converted into a numerical representation with a $4 * 4$ matrix.

So, we already had 4 samples here. So, there are 4 samples in the transform matrix and we have now 4 columns each column corresponds to 1 genre, for example, the first column corresponds to genre action. So, wherever there is action in the first third, and 4th sample we see value 1. Then the second column corresponds to the comedy genre and wherever

there is a comedy in a genre which is in the first and second sample we have values 1 over there.

The third one corresponds to science fiction and science fiction is only present in the last sample. So, you have value 1 corresponding to science fiction in the last sample and the third column. And the final column corresponds to thriller the thriller genre is associated with the third and 4th samples. So, you will see values 1 for the third and 4th samples in the last column.

(Refer Slide Time: 13:13)



So, that was about encoding categorical features as well as labels we have another important transformation that we need to perform, and most often we have to add a dummy feature to our dataset because the dummy feature is not present and the dummy feature corresponds to bias and sklearn, unfortunately, provides us a class called add dummy feature and we can use that add dummy feature to augment the dataset with a column vector where each value in that vector is 1.

Let us take a complete example. Let us say this is our original feature vector with 4 samples and 2 features and if we call add dummy feature on this we get an additional column with all ones in it. So, this is the dummy feature that is added. So, that's it from the categorical encoders, and in this video, we looked at how to encode the categorical features as well as labels in a numerical fashion using various categorical encoders provided by sklearn.