

Feature Encoding

Another very important check before feeding data to Machine Learning Algorithms is to check whether all the features are of Numerical data types or not. If the dataset has categorical features with String data types, then the machine cannot interpret it directly, thus it needs to be converted into numerical fields. This process of converting the String (or object) Data type features to numerical data type features is termed as encoding.

Categorical features are classified into two categories –

1. Nominal categorical variables

Variables that are just names and are used for labeling/distinction between multiple classes. In addition, there does not exist any order among them, and none of them have any numerical significance.

Few examples could be Cities, Gender, Claim (Yes/No) etc.

2. Ordinal categorical variables

Variables in which the value of the data is captured from an ordered set i.e. there exist an inherent order among the elements.

For example: Customer feedback survey on a Likert scale with different levels such as Poor, Fair, Good, and Very Good.

Once we classify the categorical features present in our dataset, next step is to choose an appropriate encoding technique.

Three most commonly used Encoding techniques -

1. One-Hot Encoding :

In this technique, for each category of a feature, we create a new column (sometimes called a dummy variable) with binary encoding (0 or 1) to denote whether a particular row belongs to this category.

For example, a feature with name of the States would be encoded as below –

State	State_Maharashtra	State_Tamil Nadu	State_Delhi	State_Karnataka	State_Gujarat	State_Uttar Pradesh
Maharashtra	1	0	0	0	0	0
Tamil Nadu	0	1	0	0	0	0
Delhi	0	0	1	0	0	0
Karnataka	0	0	0	1	0	0
Gujarat	0	0	0	0	1	0
Uttar Pradesh	0	0	0	0	0	1

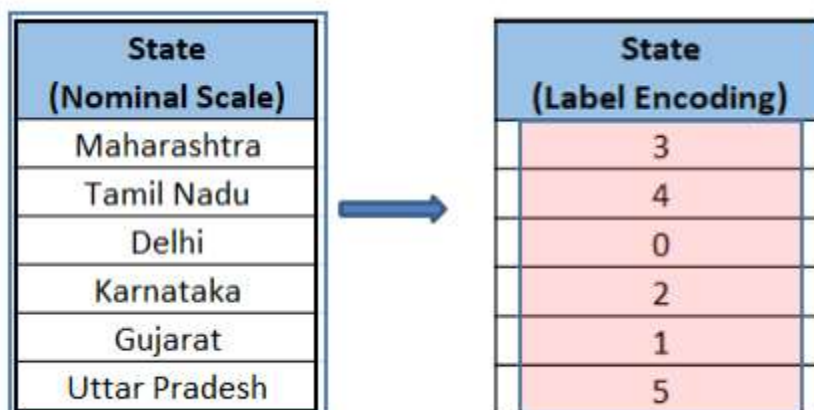
A potential drawback of this method is a significant increase in the dimensionality of the dataset (which is called a Curse of Dimensionality). Thus, one-hot encoding is mostly used for features with low number of unique values.

In addition, creating dummy variables introduces a form of redundancy to the dataset. If a feature has six categories like in the example shown above, we only need to have five dummy variables because, if an observation is neither of the five, it must be the sixth one. If the redundant dummy variable is not removed, it leads to Multicollinearity issue in the newly created dummy variables i.e. the independent variables have a relationship among themselves. Presence of Multicollinearity fails the assumptions of some of the parametric models such as Linear Regression and Logistic Regression, leading to statistically insignificant results.

2. Label Encoding:

In label encoding, we replace the categorical value with a numeric value between 0 and the number of classes minus 1. If the categorical variable value contains 5 distinct classes, we use (0, 1, 2, 3, and 4) as the labels.

Using the same example as before, below is what the State feature would be encoded into using label encoding –



State (Nominal Scale)
Maharashtra
Tamil Nadu
Delhi
Karnataka
Gujarat
Uttar Pradesh

State (Label Encoding)
3
4
0
2
1
5

The labels (0 to 5) are assigned in the Alphabetical order of the unique values i.e. Delhi:0, Gujarat:1 and so on.

3. Ordinal Encoding

An Ordinal Encoder is used to encode categorical features into an ordinal numerical value (ordered set). This approach transforms categorical value to numerical value in ordered sets. This encoding technique appears almost similar to Label Encoding. However, label encoding would not consider whether a variable is ordinal or not, but in the case of ordinal encoding, it will assign a sequence of numerical values as per the order of data.

For example, for a typical feedback feature the results of the Ordinal encoding would be as follows –

Original Encoding	Ordinal Encoding
Poor	1
Good	2
Very Good	3
Excellent	4

Note that if we were to use Label Encoding instead, Excellent would be assigned 1, Good 2 and so on, leading to misinformation regarding the order.

Python Codes -

1. One-hot Encoding :

i) Importing the encoders from sklearn

```
from sklearn.preprocessing import OneHotEncoder
```

```
# Initializing the encoder object
```

```
# drop = 'first' is used to handle the Multicollinearity issue
```

```
# sparse = False is used to ensure that the data is not stored in a Sparse matrix format.
```

```
one_hot = OneHotEncoder(drop = 'first', sparse=False)
```

```
# Fitting the encoder object on the train set
```

```
one_hot.fit(data_train[['referral_preferred_city']])
```

```
# Transforming the Train and Test sets to be used for modelling
```

```
data_train_onehot = one_hot.transform(data_train[['referral_preferred_city']])
```

```
data_test_onehot = one_hot.transform(data_test[['referral_preferred_city']])
```

```
# Inverse Transforming (Decoding) to get original labels
```

```
one_hot.inverse_transform([[1,0,0],[0,1,0],[0,0,1],[0,0,0]])
```

ii) Using Pandas Get_dummies –

Argument drop_first = True to handle the issue of Multicollinearity

```
data_one_hot = pd.get_dummies(data['referral_preferred_city'],drop_first=True)
```

2. Label Encoding –

Argument Categories = None to use the unique classes from the feature, and
ordered = None to ensure nominal encoding.

```
data['ref_city_cat'] = pd.Categorical(values =data['referral_preferred_city'] ,  
categories=None, ordered=None)
```

3. Ordinal Encoding –

i) Importing the encoders from sklearn

```
from sklearn.preprocessing import OrdinalEncoder
```

Initializing the encoder object

Passing the predefined categories as argument to ensure a required order

```
ord_encoder = OrdinalEncoder(categories=[['Pune','Delhi','Mumbai','Bangalore']])
```

Fitting the encoder object on the train set

```
ord_encoder.fit(data_train[['referral_preferred_city']])
```

Transforming the Train and Test sets to be used for modelling

```
data_train['referral_preferred_city_enc'] = ord_encoder.transform(data_train  
[['referral_preferred_city']]) .reshape(1,-1)[0]
```

```
data_test ['referral_preferred_city_enc'] = ord_encoder.transform(data_test  
[['referral_preferred_city']]) .reshape(1,-1)[0]
```

#Inverse Transforming (Decoding) to get original labels

```
ord_encoder.inverse_transform([[0],[1],[2],[3]])
```

ii) Using Pandas Categorical function –

#Argument categories to define the custom classes, and ordered = True to achieve an ordered set (Ordinal Encoding)

```
data['ref_city_cat']= pd.Categorical(values =data['referral_preferred_city'] ,  
categories=['Mumbai','Delhi','Bangalore','Pune'],  
ordered=True)
```