DATA ENCODING

16 October 2023 13:18

Data Encoding is an important pre-processing step in Machine Learning. It refers to the process of converting categorical or textual data into numerical format, so that it can be used as input for algorithms to process.

What is Categorical Data?
When we collect data, we often encounter different types of variables. One such type is categorical variables. Categorical variables are usually represented as 'strings' or 'categories' and are finite in number.
There are two types of categorical data -

• ***Ordinal Data:*** *Rating,Positioning[1st,2nd,3rd],Shirt size(38,40,41) . Where Number is related to Order*

• ***Nominal Data:***
In nominal data, the values are distinct and do not have any inherent order or numerical significance.
Ex: colour of hair[Brown,Blonde,Red etc] ,True or False

Experience(years)	Education	Salary
4	BE	80K
6	PHD	100K
3	MASTER	80K
6	PHD	120K

Categorical features

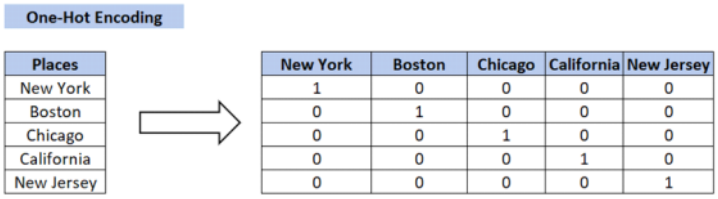
Data Encoding Aim ⟶ To convert Categorical Features into ⟶ Suitable Numerical Feature so that ML model can be Trained.

1) NOMINAL/ONE HOT ENCODING:

Nominal Encoding is a technique used to Transform Variable that have no intrinsic Ordering into Numerical Values that can be used in ML Models. One common method for nominal encoding is one hot encoding which creates a binary vector for each category in the variable.

A Binary Column is created for each Unique Category in the variable.
• If a category is present in a sample, the corresponding column is set to 1, and all other columns are set to 0.
• For example, if a variable has three categories 'A', 'B' and 'C', three columns will be created and a sample with category 'B' will have the value [0,1,0].

one-hot encoding, for N categories in a variable, it uses N binary variables.




```
# One-Hot Encoding:
# create a sample dataframe with a categorical variable
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red']})

# perform one-hot encoding on the 'color' column
one_hot = pd.get_dummies(df['color'])

# concatenate the one-hot encoding with the original dataframe
df1 = pd.concat([df, one_hot], axis=1)

# drop the original 'color' column
df1 = df1.drop('color', axis=1)
```

Disadvantage of OHE:
Sparse Matrix: For 10 locations it will create 10 columns and assign 1,0 that will lead to overfitting.
If you have more categories than we don't use OHE it will Increase Features.

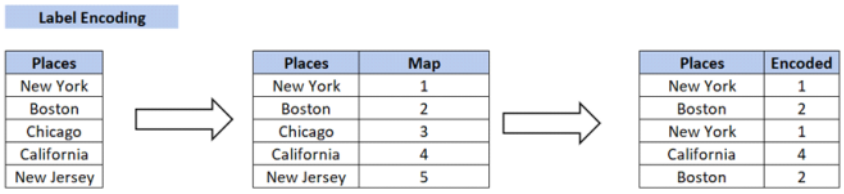
2) LABEL ENCODING:

IT is named as label in label encoding because it will assign certain fixed value to that feature whether it will repeat it will assign the same value to it.

Label Encoding:

Each unique category is assigned a Unique Integer value.

This is a simpler encoding method, but it has a Drawback in that the assigned integers may be misinterpreted by the machine learning algorithm as having an Ordered Relationship when in fact they do not.




```
from sklearn.preprocessing import LabelEncoder

# Create a sample dataframe with categorical data
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red', 'green']})

print(f"Before Encoding the Data:\n\n{df}\n")

# Create a LabelEncoder object
le = LabelEncoder()

# Fit and transform the categorical data
df['color_label'] = le.fit_transform(df['color'])
```

Advantages of Label Encoding:

Simplicity: Label encoding is straightforward to implement and can quickly convert categorical data into numerical form.

Compatibility: Many machine learning algorithms and libraries accept numerical inputs, making label encoding a convenient way to prepare data for these algorithms.

Disadvantages of Label Encoding:

Implicit Order: Label encoding can introduce an implicit ordinal relationship between the categories, even when no such relationship exists in the original data. This can mislead the model into interpreting the data with a false sense of order.

One major disadvantage of Label encoder is that it will compare that green>blue>red and that lead to misinterpretation

Sensitivity to Large Values: If the encoded values are significantly large, some algorithms may treat them as continuous values, leading to incorrect interpretations and potentially affecting the model's performance.

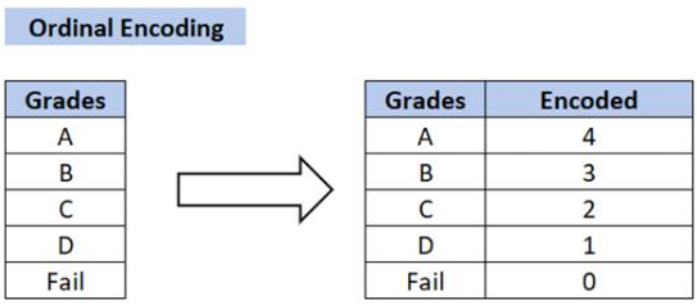
1) ORDINAL ENCODING: (Categories With Rank)

To Counter the Issue of Label Encoding we have Ordinal Encoding.

Ordinal Encoding is used when the categories in a variable have a Natural Ordering (into Numerical).

• In this method, the categories are assigned a numerical value based on their order, such as 1, 2, 3, etc.

For example, if a variable has categories ‘Low’, ‘Medium’ and ‘High’, they can be assigned the values 1, 2, and 3, respectively.



```
# Ordinal Encoding:
# create a sample dataframe with a categorical variable
df = pd.DataFrame({'quality': ['low', 'medium', 'high', 'medium']})
print(f"Before Encoding the Data:\n\n{df}\n")

# specify the order of the categories
quality_map = {'low': 0, 'medium': 1, 'high': 2}

# perform ordinal encoding on the 'quality' column
df['quality_map'] = df['quality'].map(quality_map)
```

Advantages of Ordinal Encoding:

Preserves Order Information: Ordinal encoding retains the ordinal relationship among the categories, making it suitable for variables where the categories have a clear and meaningful order.

Reduces Dimensionality: Ordinal encoding reduces the dimensionality of the data compared to one-hot encoding, making it more memory-efficient and suitable for models that might struggle with high-dimensional data.

Disadvantages of Ordinal Encoding:

Assumed Equidistance: Ordinal encoding assumes an equal distance between the categories, which might not always hold true. Treating categories as equidistant may introduce biases and inaccuracies in the data representation.