4-ordinal_encoder

October 20, 2024

1 Ordinal Encoder

is a technique used to convert categorical features into numerical values, where the categories have a clear order or ranking. Unlike one-hot encoding, which treats all categories as independent, ordinal encoding assigns each unique category an integer value based on its rank or order.

Why Ordinal Encoding is Important:

- Ordered Categories: It is specifically used for ordinal data, where the categories have a meaningful order or ranking, but the distances between these categories might not be uniform.
- Simpler Representation: It provides a simpler, more compact representation of categorical features compared to one-hot encoding, especially when the feature has an inherent order and a small number of categories.

When to Use Ordinal Encoding:

- Ordinal Categorical Data: Use ordinal encoding when the categorical feature has an intrinsic order. For example, "low," "medium," and "high" represent ordered categories, but one-hot encoding would not capture the ranking.
- When Preserving Order is Important: If the relationship between categories is important for the model to understand, ordinal encoding helps preserve this information. Smaller Cardinality: When dealing with ordinal features with a small number of categories, ordinal encoding is efficient and works well with most machine learning algorithms.

How Ordinal Encoding Works: Let's take an example where we have a categorical feature "Quality" with values: "Low," "Medium," and "High." Ordinal encoding assigns each category a numerical value based on the order:

Quality Low 0 Medium 1 High 2

Color	Ordinal Encoding
Low	0
Medium	1
High	2

In this case, "Low" is mapped to 0, "Medium" to 1, and "High" to 2, which captures the inherent ranking of the categories.

When Not to Use Ordinal Encoding:

- Non-Ordinal Categorical Data: If the categorical feature has no inherent order (e.g., “color” or “country”), ordinal encoding may mislead the model because it implies a relationship or ranking between the categories that doesn’t exist.
- Assumed Numerical Relationships: Some machine learning models, like linear regression, may interpret ordinal values as having linear relationships (i.e., assuming that the difference between “Medium” and “Low” is the same as between “High” and “Medium”), which is not always correct.

Use Cases of Ordinal Encoding:

- Education Levels: Categories such as “Primary,” “Secondary,” and “Tertiary” have an inherent order, making them a perfect candidate for ordinal encoding.
- Customer Satisfaction Levels: Responses like “Very Dissatisfied,” “Dissatisfied,” “Neutral,” “Satisfied,” and “Very Satisfied” can be encoded in their natural order.
- Rating Scales: Ratings such as “Low,” “Medium,” and “High” in product reviews can be ordinal encoded since they imply a ranking.

Summary:

- Ordinal Encoding converts categorical data into numerical data, preserving the order of categories.
- It is used when the categorical variable has an inherent order (e.g., “low,” “medium,” “high”).
- It is compact and efficient but may lead to misleading interpretations if used on non-ordinal categorical data.
- Some algorithms (e.g., decision trees, random forests) work well with ordinal encoding, but others (e.g., linear models) may misinterpret the encoding.

[1]:
```
import pandas as pd

d = {'sales':␣
 ↪[100000,222000,1000000,522000,111111,222222,1111111,20000,75000,90000,1000000,10000],␣
 ↪'city':␣
 ↪['Tampa','Tampa','Orlando','Jacksonville','Miami','Jacksonville','Miami','Miami','Orlando',
 ↪'size': ['Small',␣
 ↪'Medium','Large','Large','Small','Medium','Large','Small','Medium','Medium','Medium','Small

df = pd.DataFrame(data=d)
df.head()
```

[1]:
```
     sales          city    size
0   100000         Tampa   Small
1   222000         Tampa  Medium
2  1000000       Orlando   Large
3   522000  Jacksonville   Large
```

```
4   111111         Miami    Small
```

[2]:

```
df['size'].unique()
sizes = ['Small', 'Medium', 'Large']
```

[3]:

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder(categories = [sizes])
size_enc = enc.fit_transform(df[['size']])
size_enc
```

[3]:

```
array([[0.],
       [1.],
       [2.],
       [2.],
       [0.],
       [1.],
       [2.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.]])
```

[4]:

```
df['size_enc'] = size_enc
df
```

[4]:

```
      sales          city    size  size_enc
0    100000         Tampa   Small       0.0
1    222000         Tampa  Medium       1.0
2   1000000       Orlando   Large       2.0
3    522000  Jacksonville   Large       2.0
4    111111         Miami   Small       0.0
5    222222  Jacksonville  Medium       1.0
6   1111111         Miami   Large       2.0
7     20000         Miami   Small       0.0
8     75000       Orlando  Medium       1.0
9     90000       Orlando  Medium       1.0
10  1000000       Orlando  Medium       1.0
11    10000       Orlando   Small       0.0
```