Course: Data Science Tools and Techniques

Data Preprocessing

Dr. Safdar Ali

Explore and discuss the process of data cleaning, with understanding of its importance, common challenges, and effective techniques along with data transformation.

Encoding Categorical Data

What is Categorical Data?

- Categorical data refers to any kind of non-numeric data such as names, labels, etc.

- It is discrete values (e.g., colors, labels, categories).

- Two types:

  - Nominal: No order (e.g., colors, country names)

  - Ordinal: Has order (e.g., low, medium, high)

Encoding Categorical Data

- It is a process in which categorical data transform into a numerical format.

- Several techniques of encoding categorical data are exist.

- Choice is often depend on the nature of the categorical data and the machine learning model.

Most common techniques

- **Label Encoding**: Assigns numeric values to categories.

- It is best for ordinal data with a natural order.

- It is not good for nominal data (where there is no useful order). .e.g., encoding "Red", "Blue", and "Green" as 0, 1, and 2 implies an ordinal relationship that doesn't exist.

- **One-Hot Encoding**: Creates binary columns for each category.
- It is ideal for nominal data (without order) with relatively few categories.

- It increases the dimensionality of the dataset, which could become problematic for many categories..

Most common techniques

- **Binary Encoding:** Suitable for high cardinality features (many unique categories).
- It is more efficient than one-hot encoding for datasets with many categories.
- Binary encoding might not offer much improvement over other techniques for datasets with few categories.

- **Target Encoding**: Replaces categories with mean of target variable (used in ML). It is effective for numerical relationships with the target variable, particularly in regression tasks.

- This encoding can lead to data leakage if not handled properly, particularly if the encoding is done before splitting the dataset into training and testing sets.

Most common techniques

- **Frequency Encoding:** Useful when categories have significantly different frequencies in the dataset.
- This type of encoding may introduce bias, especially if a particular category appears much more often than others, skewing the model's interpretation.

Each encoding technique has its own merit and demerit. Choice of encoding depends on the data type, model requirements, and specific features of dataset.

Label Encoding

- It is the simplest form of encoding where each unique category is assigned an integer value.
- It works well when the categorical data has an inherent order (ordinal data), such as "low", "medium", and "high".

Example:

If a dataset have a "Size" column with three categories:

"Small"

"Medium"

"Large"

Label Encoding would assign:

Small → 0

Medium → 1

Large → 2

```
from sklearn.preprocessing import LabelEncoder

data = ['Small', 'Medium', 'Large', 'Small', 'Medium']
encoder = LabelEncoder()
encoded_data = encoder.fit_transform(data)
print(encoded_data)
```

Output:

```
[0 1 2 0 1]
```

```
from sklearn.preprocessing import LabelEncoder

def label_encode(data):
  """Encodes categorical data using Label Encoding.
    Args:
    data (list): A list of categorical values.
    Returns:
    list: A list of encoded integer values.
  """
  encoder = LabelEncoder()
# Initialize the Label Encoder
  encoded_data = encoder.fit_transform(data)
# Fit and transform the data

return encoded_data

# Example usage
data = ['Red', 'Blue', 'Green', 'Red', 'Green']
print(label_encode(data))
# Output: [2, 0, 1, 2, 1]
```

One-Hot Encoding

- It creates a new binary column for each category and assigns a 1 or 0, depending on whether the category is present or not.
- It is often used for nominal data where there is no order.

Example:

For the "Color" column:

Red

Blue

Green

One-Hot Encoding would transform the data into:

Color	Red	Blue	Green
Red	1	0	0
Blue	0	1	0
Green	0	0	1
Red	1	0	0

```
import pandas as pd

data = ['Red', 'Blue', 'Green', 'Red']

df = pd.DataFrame(data,
columns=['Color'])

encoded_df = pd.get_dummies(df,
columns=['Color'])

print(encoded_df)
```

Output:

```
   Color_Blue  Color_Green  Color_Red
0      0           0            1
1      1           0            0
2      0           1            0
3      0           0            1
```

Binary Encoding

- It is a compromise between label encoding and one-hot encoding. It first assigns an **integer value** to each category and then converts those integers into **binary numbers**.
- It reduces the dimensionality compared to one-hot encoding, especially for high cardinality variables.

Example: For a "Category" column with four categories: A, B, C, D

Binary Encoding would assign:

A → 00
B → 01
C → 10
D → 11

data is encoded as:

Category	Binary Encoding
A	00
B	01
C	10
D	11

```
import category_encoders as ce
data = ['A', 'B', 'C', 'D']
encoder = ce.BinaryEncoder(cols=['Category'])
df = pd.DataFrame(data, columns=['Category'])
encoded_df = encoder.fit_transform(df)
print(encoded_df)
```

Output:

```
   Category_0  Category_1
0  0           0
1  0           1
2  1           0
3  1           1
```

Steps for Binary Encoding

Step-1: Assign unique integer (starting from 0) to categories

Step-2: Convert these integers to binary form

Step-3: Create columns (consider bits for the largest integer) and each binary digit is stored in a separate column.

Example: Let's consider 6 unique Color categories of **Red, Blue, Green, Yellow, Black, White**

Step 1: Assigning unique integer to Categories

- Red -> 0
- Blue -> 1
- Green -> 2
- Yellow -> 3
- Black -> 4
- White -> 5

Step 2: Convert Numbers to Binary

The largest number is 5 and convert each integers to binary:

- Red (0) → 000
- Blue (1) → 001
- Green (2) → 010
- Yellow (3) → 011
- Black (4) → 100
- White (5) → 101

Step 3: Creating Columns

Color	Bit 1	Bit 2	Bit 3
Red	0	0	0
Blue	0	0	1
Green	0	1	0
Yellow	0	1	1
Black	1	0	0
White	1	0	1

Target Encoding (Mean Encoding)

- It involves encoding categories based on the mean of the target variable.
- It is often used for ordinal or high-cardinality features in regression tasks.

Example:

Let's consider a dataset with a "City" column and a target variable "Income".

City	Income
Paris	60,000
London	70,000
Paris	65,000
London	80,000

Replace each city with the average income for that city:

- Paris → (60,000 + 65,000) / 2 = 62,500
- London → (70,000 + 80,000) / 2 = 75,000

the dataset would become:

City	Income	Encoded City
Paris	60,000	62,500
London	70,000	75,000
Paris	65,000	62,500
London	80,000	75,000

```python
import pandas as pd

data = {'City': ['Paris', 'London', 'Paris', 'London'],
'Income': [60000, 70000, 65000, 80000]}

df = pd.DataFrame(data)

mean_encoded_city = df.groupby('City')['Income'].mean()
# Compute mean for each category

df['Encoded City'] = df['City'].map(mean_encoded_city)
# Map mean values to categories

print(df)
```

Output:

```
          City          Income
           Encoded City
0   Paris          60000        62500
1  London          70000        75000
2   Paris          65000        62500
3  London          80000        75000
```

Frequency/Count Encoding

- In this encoding, each category is replaced by its frequency.
- Best when there is a significant difference in the frequency of categories.

Example:

Consider a "Fruit" column:

Apple (appears 3 times)

Banana (appears 2 times)

Cherry (appears 1 time)

replace:

Apple → 3

Banana → 2

Cherry → 1

```
data = ['Apple', 'Banana', 'Apple', 'Cherry', 'Banana',
'Apple']
df = pd.DataFrame(data, columns=['Fruit’])
# Convert to DataFrame
frequency_encoding = df['Fruit'].value_counts()
 # Compute category frequency
df['Encoded Fruit'] = df['Fruit'].map(frequency_encoding)
# Map frequencies
print(df)
```

Output:

	Fruit	Encoded Fruit
0	Apple	3
1	Banana	2
2	Apple	3
3	Cherry	1
4	Banana	2
5	Apple	3

Use: Huffman coding assigns shorter codes to more frequent characters and longer codes to less frequent ones, reducing file sizes in formats like ZIP and MP3

Automating Data Preprocessing Pipelines

Automating Data Preprocessing Pipelines?

- Automating repetitive preprocessing steps to improve efficiency.
- Ensured consistency in data preparation
- Reduced manual intervention and human errors
- Faster model development
- Easily adaptable for new data sources
- Scalable for large datasets and machine learning workflows

Main Stages in Data Preprocessing Pipelines

- Handling missing values, etc.
- Encoding categorical data
- Feature scaling (normalization, standardization)
- Feature engineering
- Splitting data into training/testing sets
- Automating above stages using Python libraries.

Python Code for Automating Preprocessing with Pipelines

```
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import
StandardScaler, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.compose import
ColumnTransformer

import pandas as pd
```

```
# Sample Data
df = pd.DataFrame({'Age': [25, np.nan,
30, 35], 'City': ['NY', 'LA', 'SF', 'NY']})
# Define Transformers
num_pipeline = Pipeline([('imputer',
SimpleImputer(strategy='mean')),
('scaler', StandardScaler())
])
cat_pipeline = Pipeline([('imputer',
SimpleImputer(strategy='most_frequent'))
, ('encoder', OneHotEncoder())
])
# Combine Pipelines
preprocessor = ColumnTransformer([
    ('num', num_pipeline, ['Age']),
    ('cat', cat_pipeline, ['City'])
])
# Apply Transformation
df_transformed =
preprocessor.fit_transform(df)
print(df_transformed)
```

Tracking Pipelines with MLflow

- Track preprocessing steps & experiments using MLflow.
- Allows reproducibility in machine learning workflows.
- Manage versions of preprocessing pipelines.

```
import mlflow
import mlflow.sklearn

# Start an MLflow experiment
mlflow.start_run()

# Log preprocessing pipeline
mlflow.sklearn.log_model(preprocessor,
'preprocessing_pipeline')
mlflow.end_run()
```

So for, we have discussed and practiced:

Data Transformation and Common Format Conversion such as:

- **Aggregation (Summarizing data)**
- **Reshaping (Pivoting, Melting)**
- **Scaling (Normalization, Standardization)**

- **Data Visualization**

- **Encoding Categorical Data**

- **Data Preprocessing Automation**

This practice enhanced the analytical capabilities to handle complex datasets using powerful tools for data Preprocessing, computation and statistical analysis.