PW SKILLS

Lesson Plan

Data Encoding & Nominal Encoding

Topics Covered

- Introduction to Data Encoding
- Types of Data in Machine Learning
- Understanding Nominal Encoding
- Common Nominal Encoding Techniques
  - One Hot Encoding
  - Label Encoding
  - Binary Encoding
- Practical Implementation

Introduction to Data Encoding

- Data encoding refers to the process of converting categorical or qualitative data into a numerical format that can be easily processed by machine learning algorithms.
- It plays a crucial role in preparing and structuring data for analysis and model training.
- Many machine learning algorithms, especially those based on mathematical equations, require numerical input.
- Data encoding enables algorithms to process and make sense of categorical features, which are prevalent in real-world datasets.
- Proper encoding improves the performance and accuracy of machine learning models.
- Introduce the main types of data encoding: Nominal Encoding, Ordinal Encoding, and Binary Encoding.
- Mention that the choice of encoding method depends on the nature of the data and the requirements of the machine learning algorithm.
- Highlight that this discussion will focus on Nominal Encoding and its significance in handling categorical variables.

Types of Data in Machine Learning

- **Categorical Data vs. Numerical Data:**
  - Define categorical data as variables that can take on a limited and fixed number of values or categories.
  - Define numerical data as variables that represent measurable quantities and can take on a range of values.
  - Explain that many real-world datasets contain categorical variables, such as gender, color, or product type.
  - Emphasize that without proper handling, machine learning models may misinterpret categorical data, leading to inaccurate predictions or classifications.

Understanding Nominal Encoding

- Define nominal encoding as a technique used to represent categorical data where no order or ranking is implied among the categories.
- Emphasize that nominal encoding transforms categories into unique numerical representations, allowing machine learning algorithms to understand and process them effectively.

- Provide examples of nominal variables, such as country names, colors, or animal types.
- Illustrate the need for encoding by explaining how these variables cannot be directly used in many machine learning algorithms.
- Address the issue of dimensionality increase when using certain encoding methods and its impact on computational resources.

Common Nominal Encoding Techniques

- **One-Hot Encoding:**
  - One-hot encoding is a technique used to convert categorical variables into a binary matrix format. Each category is represented by a binary vector where only one bit is 1 (hot) and the rest are 0 (cold).
  - For example, if you have a categorical variable "Color" with values ["Red", "Blue", "Green"], after one-hot encoding, each color becomes a binary vector such as [1, 0, 0], [0, 1, 0], and [0, 0, 1] respectively.

  - **Pros**:
    - Preserves all information about the categorical variable.
    - Works well with algorithms that do not assume any ordinal relationship between categories.
    - Easy to implement in most machine learning frameworks.

  - **Cons:**
    - This can lead to a high-dimensional sparse matrix, especially if the categorical variable has many unique categories.
    - May introduce multicollinearity in linear models if not handled properly.
    - Increases the computational complexity and memory requirement.

  - **Examples:**
    - Sentiment analysis where categorical variables may include emotions like "happy", "sad", or "neutral".
    - Customer segmentation based on demographics such as age groups, gender, and location.
    - Image classification tasks where labels represent different objects or classes.

- **Label Encoding:**
  - Label encoding is a technique used to convert categorical variables into numerical format by assigning a unique integer to each category.
  - Each category is replaced by its corresponding integer label. For example, ["Red", "Blue", "Green"] may be encoded as [0, 1, 2].
  - Label encoding is suitable when there is an ordinal relationship among the categories. It works well for categorical variables where the categories have a natural order or hierarchy.
  - For instance, "low", "medium", and "high" can be encoded as [0, 1, 2] respectively.

- **Binary Encoding:**
  - Binary encoding is a technique that represents categorical variables as binary numbers.
  - Each category is first encoded as an integer, and then the integer is converted into binary digits.
  - These binary digits are used as features. For example, if a categorical variable "Color" with values ["Red", "Blue", "Green"] is binary encoded, it may result in binary representations such as [00, 01, 10].
  - Reduces dimensionality compared to one-hot encoding while still preserving information.
  - Can be more memory-efficient, especially for high cardinality categorical variables.
  - Works well with algorithms that cannot handle high-dimensional sparse matrices efficiently.
  - Helps in capturing relationships between categories in a more compact representation.
  - Binary encoding is often compared with one-hot encoding and label encoding. Unlike one-hot encoding, binary encoding results in fewer dimensions, making it computationally more efficient.
  - Compared to label encoding, binary encoding does not impose any ordinality assumptions on the categorical variables, making it more suitable for non-ordinal categorical data.

Practical Implementation

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Load the Iris dataset
iris = load_iris()
data = iris.data
target = iris.target

# Create a DataFrame from the dataset
columns = [f"feature_{i}" for i in range(data.shape[1])]
df = pd.DataFrame(data, columns=columns)
df["target"] = target

# Display the original DataFrame
print("Original DataFrame:")
print(df.head())

# Label Encoding for the 'target' column
le = LabelEncoder()
df['target_encoded'] = le.fit_transform(df['target'])

# Display the DataFrame after Label Encoding
print("\nDataFrame after Label Encoding:")
print(df.head())

# One-Hot Encoding for the 'target' column
ohe = OneHotEncoder(sparse=False)
ohe_result = ohe.fit_transform(df[['target']])

# Create a new DataFrame for One-Hot Encoding results
ohe_df = pd.DataFrame(ohe_result, columns=[f"target_{i}" for i in
range(ohe_result.shape[1])])

# Concatenate the One-Hot Encoding DataFrame with the original
DataFrame
df = pd.concat([df, ohe_df], axis=1)

# Display the DataFrame after One-Hot Encoding
print("\nDataFrame after One-Hot Encoding:")
print(df.head())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(df.drop(['target', 'target_encoded'] +
ohe_df.columns.tolist(), axis=1),
                                                    df['target_en
coded'], test_size=0.2, random_state=42)

# Display the training and testing sets
print("\nTraining Set:")
print(X_train.head())
print("\nTesting Set:")
print(X_test.head())
```

Output:

DataFrame after One-Hot Encoding:

```
feature_0  feature_1  feature_2  feature_3  target  target_encoded  \
0        5.1        3.5        1.4        0.2       0               0
1        4.9        3.0        1.4        0.2       0               0
2        4.7        3.2        1.3        0.2       0               0
3        4.6        3.1        1.5        0.2       0               0
4        5.0        3.6        1.4        0.2       0               0

target_0  target_1  target_2
0       1.0       0.0       0.0
1       1.0       0.0       0.0
2       1.0       0.0       0.0
3       1.0       0.0       0.0
4       1.0       0.0       0.0
```

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(

Training Set:

```
feature_0  feature_1  feature_2  feature_3
22         4.6        3.6        1.0        0.2
15         5.7        4.4        1.5        0.4
65         6.7        3.1        4.4        1.4
11         4.8        3.4        1.6        0.2
42         4.4        3.2        1.3        0.2
```

Testing Set:

```
feature_0  feature_1  feature_2  feature_3
73         6.1        2.8        4.7        1.2
18         5.7        3.8        1.7        0.3
118        7.7        2.6        6.9        2.3
78         6.0        2.9        4.5        1.5
76         6.8        2.8        4.8        1.4
```