feature-engineering-3-4

August 13, 2023

```
[ ]: # Q1. What is data encoding? How is it useful in data science?
```

Data encoding is the process of converting data from one format or representation into another format, often to facilitate storage, transmission, processing, or interpretation. In the context of data science, encoding is particularly relevant when dealing with categorical data, text data, or any data that isn't in a suitable numerical format for analysis or modeling.

Here are some common scenarios where data encoding is useful in data science:

1 Categorical Variable Encoding: Categorical variables represent distinct categories or labels, such as colors, types of products, or countries. Machine learning algorithms generally require numerical input, so categorical variables need to be encoded into numerical values. Common encoding techniques include one-hot encoding, label encoding, and ordinal encoding.

2 Text Data Processing: Textual data is often encoded into numerical representations to be used in machine learning models. Techniques like bag-of-words, TF-IDF (Term Frequency-Inverse Document Frequency), and word embeddings (e.g., Word2Vec, GloVe) convert text into numerical vectors that capture semantic information.

3 Feature Engineering: Data encoding is often a part of feature engineering, where existing features are transformed or combined to create new features that might have more meaningful information for the model.

4 Preprocessing for Machine Learning: Before training a machine learning model, it's essential to preprocess the data, which can include encoding categorical variables, normalizing numerical variables, handling missing values, and more.

5 Data Compression: Encoding techniques can also be used for data compression, where the data is transformed into a more compact representation, reducing storage requirements while maintaining important information.

6 Data Transmission: Encoding is crucial when transmitting data over networks or storing data in databases. It ensures data integrity and efficient use of storage or bandwidth.

7 NLP and Language Processing: In natural language processing (NLP), data encoding is fundamental to processing and understanding human language. Text must be encoded in a way that captures its meaning for language-related tasks.

8 Time Series Data: When dealing with time series data, encoding timestamps into suitable formats allows for meaningful temporal analysis and modeling.

```
[ ]: # Q2. What is nominal encoding? Provide an example of how you would use it in a␣
     ↪real-world scenario
```

Nominal encoding is a method of converting categorical data where the categories have no inherent order or ranking into numerical values. Nominal encoding is necessary when working with machine learning algorithms that require numerical input, as these algorithms can't directly handle categorical labels.

There are a few common techniques for nominal encoding:

9 One-Hot Encoding: In one-hot encoding, each category is represented as a binary vector where each element corresponds to a category. Only one element in the vector is 1 (indicating the category), and the rest are 0. This method creates multiple binary columns, one for each category.

10 Label Encoding: Label encoding assigns a unique integer value to each category. However, this method should be used with caution for nominal data because it can mistakenly introduce ordinal relationships that don't actually exist.

Here's an example of nominal encoding in a real-world scenario:

Scenario: Imagine you are working on a classification problem to predict whether an email is "spam" or "not spam" based on certain features. One of the features is the email's "country of origin."

Possible values: "USA", "Canada", "UK", "Australia", "France"

In this case, you can use one-hot encoding to convert the categorical feature "Country of Origin" into numerical format:

Country_USA Country_Canada Country_UK Country_Australia Country_France 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0

Each email's "Country of Origin" is now represented by a binary vector indicating the presence or absence of each country. This transformation allows machine learning algorithms to understand and use the "Country of Origin" feature effectively.

One-hot encoding is particularly useful in this scenario because it correctly captures the nominal nature of the data without introducing unintended ordinal relationships.

```
[ ]: # Q3. In what situations is nominal encoding preferred over one-hot encoding?␣
     ↪Provide a practical example
```

Nominal encoding is typically preferred over one-hot encoding in situations where the categorical feature has a large number of distinct categories, and the resulting one-hot encoded representation would lead to a significant increase in the number of features.

One-hot encoding can result in a sparse matrix with many binary columns, which might introduce computational inefficiencies, increase memory usage, and potentially lead to overfitting in some cases.

Here's a practical example where nominal encoding might be preferred over one-hot encoding:

11 Scenario: You are working on a project that involves analyzing customer behavior on an e-commerce website. One of the features is the "Product Category," and the website has a very large number of product categories, numbering in the thousands.

Possible values: "Electronics", "Clothing", "Home Decor", "Books", "Toys", "Sports Equipment" (thousands of categories)

In this case, using one-hot encoding would result in thousands of binary columns, making the dataset extremely wide.

12 Potential Overfitting: In machine learning models, having a very high number of features relative to the number of samples can lead to overfitting, where the model learns noise from the training data instead of true patterns.

Given these challenges, nominal encoding could be preferred in this scenario. Label encoding or other methods that convert categories into numerical values without creating thousands of new features could be more suitable. However, it's important to consider the nature of the data, the algorithms you're using, and the potential impact on model performance. Always validate the impact of encoding choices on your specific problem and dataset.

[ ]:
```
# Q4. Suppose you have a dataset containing categorical data with 5 unique␣
 ↪values.

# Which encoding technique would you use to transform this data into a format␣
 ↪suitable for machine learning algorithms?

# Explain why you made this choice.
```

The choice of encoding technique depends on the nature of the categorical data and the specific requirements of the machine learning algorithms you plan to use.

In the case where you have a categorical feature with 5 unique values, you have several options:

13 Label Encoding:

Label encoding assigns a unique integer value to each category. In your case, you would assign integers from 0 to 4 to the 5 unique values.

This technique is simple and can work well if there is an ordinal relationship between the categories. However, if there's no meaningful order among the categories, label encoding might introduce unintended ordinal relationships that could mislead the algorithm.

14 Nominal Encoding:

Since you have a small number of categories (5), one-hot encoding is a viable option. Each category would be transformed into a binary column, and each instance would have a single "1" in the corresponding column and "0"s in the others.

This technique is suitable when the categories are nominal and have no inherent order. It avoids introducing any ordinal relationships between the categories.

15 Choice and Rationale:

For a dataset containing categorical data with only 5 unique values, one-hot encoding would be a strong choice. It's straightforward to implement, preserves the nominal nature of the data, and avoids introducing unintended ordinal relationships. One-hot encoding will create 5 binary columns, one for each category, which can be efficiently handled by most machine learning algorithms.

[ ]:
```
# Q5. In a machine learning project, you have a dataset with 1000 rows and 5␣
↪columns.

# Two of the columns are categorical, and the remaining three columns are␣
↪numerical.

# If you were to use nominal encoding to transform the categorical data, how␣
↪many new columns would be created? Show your calculations.
```

When using nominal encoding, specifically one-hot encoding, to transform categorical data, each unique category in a column is transformed into a new binary column. If you have two categorical columns, each with a certain number of unique categories, you would create new binary columns for each unique category in those columns.

Let's assume the following for your dataset:

Column 1: Categorical feature 1 with 10 unique categories. Column 2: Categorical feature 2 with 5 unique categories. Column 3: Numerical feature. Column 4: Numerical feature. Column 5: Numerical feature.

For column 1, you would create 10 new binary columns (one for each category). For column 2, you would create 5 new binary columns (one for each category).

Therefore, the total number of new columns created due to nominal encoding would be 10 + 5 = 15.

So, if you were to use nominal encoding (one-hot encoding) to transform the categorical data in this specific scenario, you would create 15 new columns.

```
[ ]: # Q6. You are working with a dataset containing information about different␣
     ↪types of animals, including their species, habitat, and diet.

     # Which encoding technique would you use to transform the categorical data into␣
     ↪a format suitable for machine learning algorithms? Justify your answer
```

In the scenario of working with a dataset containing information about different types of animals, including their species, habitat, and diet, the choice of encoding technique depends on the nature of the categorical variables and the machine learning algorithms you plan to use.

Given that the categorical variables are "species," "habitat," and "diet," each with potentially multiple categories, and that these categories don't inherently have any ordinal relationship, the most suitable encoding technique would be one-hot encoding.

Here's how you would justify the choice of one-hot encoding in this case:

"Our dataset contains categorical variables such as 'species,' 'habitat,' and 'diet.' These categorical variables represent distinct categories without any inherent order. To transform this categorical data into a format suitable for machine learning algorithms, we would choose one-hot encoding.

One-hot encoding creates separate binary columns for each category, preserving the nominal nature of the data and ensuring that no unintended ordinal relationships are introduced. This technique is widely used for nominal categorical variables and is compatible with various machine learning algorithms."

```
[ ]: # Q7.You are working on a project that involves predicting customer churn for a␣
     ↪telecommunications company.

     # You have a dataset with 5 features, including the customer's gender, age,␣
     ↪contract type, monthly charges, and tenure.

     # Which encoding technique(s) would you use to transform the categorical data␣
     ↪into numerical data?


     # Provide a step-by-step explanation of how you would implement the encoding.
```

```
[8]: import pandas as pd


     # Since "gender" is binary (two categories: male and female), you can use label␣
     ↪encoding or a simple mapping to convert it into numerical values.

     # Assuming 'gender' column contains 'male' and 'female'

     # "Contract type" has multiple categories, and there's no inherent order.␣
     ↪One-hot encoding is suitable for this situation.

     # Age, Monthly Charges, Tenure (Numerical):
```

```
# #These features are already in numerical format, so no encoding is needed for␣
  ↪them.


# Sample dataset
data = {
    'gender': ['male', 'female', 'male', 'female', 'male'],
    'contract': ['A', 'B', 'A', 'C', 'B'],
    'age': [25, 30, 22, 40, 35],
    'monthly_charges': [50, 70, 40, 90, 60],
    'tenure': [12, 24, 6, 36, 18]
}

df = pd.DataFrame(data)

# Encode gender using label encoding
df['gender_encoded'] = df['gender'].map({'male': 0, 'female': 1})

# Apply one-hot encoding to contract column
contract_dummies = pd.get_dummies(df['contract'], prefix='contract')
df = pd.concat([df, contract_dummies], axis=1)

# Drop original gender and contract columns
df.drop(['gender', 'contract'], axis=1, inplace=True)

print(df)
```

```
   age  monthly_charges  tenure  gender_encoded  contract_A  contract_B  \
0   25               50      12               0           1           0
1   30               70      24               1           0           1
2   22               40       6               0           1           0
3   40               90      36               1           0           0
4   35               60      18               0           0           1

   contract_C
0           0
1           0
2           0
3           1
4           0
```