Encoding Techniques Comparison

Binary encoding vs one hot encoding vs dummy encoding please explain in context of data science with one typical scenario for each type of encoding and one common scenario which will illustrate how each type handles it differently and which one is better

Let’s break this down clearly and with relatable examples:

🔠 What are Binary Encoding, One-Hot Encoding, and Dummy Encoding?

These are **categorical data encoding** techniques — used to convert **non-numeric (categorical)** features into **numeric** form so that machine learning models can work with them.

🟨 1. One-Hot Encoding

- **What:** Creates **binary columns for each category**.
- **How:** Each unique category gets a separate column; a row has 1 for its category and 0 elsewhere.
- **#Columns Created:** Equals number of unique categories.

🧪 Example Scenario:

Feature: “Fruit” with values: `["Apple", "Banana", "Cherry"]`

One-hot encoding creates:

```
| Fruit_Apple | Fruit_Banana | Fruit_Cherry |
|      1      |       0      |       0      |
|      0      |       1      |       0      |
|      0      |       0      |       1      |
```

✅ When to use:

- When the number of unique categories is **small**.
- When categories **don’t have ordinal relationship** (e.g., fruits, city names).

🟦 2. Dummy Encoding (a variant of One-Hot)

- **What:** Similar to one-hot, **but drops one column** to avoid multicollinearity (especially useful in linear models).
- **#Columns Created:** One less than total categories.

🧪 Example Scenario:

Same "Fruit" example — only two columns are kept:

```python
| Fruit_Apple | Fruit_Banana |
|      1      |       0      | → Cherry is the base case (all 0s)
|      0      |       1      |
|      0      |       0      |
```

✅ When to use:

- When using models like **linear regression** or **logistic regression** that can be affected by multicollinearity.

🟩 3. Binary Encoding

- **What:** Converts categories to **integers**, then encodes those integers into **binary** format, using fewer columns.
- **#Columns Created:** `log2(n_categories)` (more efficient for high-cardinality)

🧪 Example Scenario:

Feature: "Fruit" with 5 categories

`["Apple", "Banana", "Cherry", "Date", "Elderberry"]`

Assign IDs:

Apple=1, Banana=2, Cherry=3, Date=4, Elderberry=5

Binary Encoding:

```javascript
| Fruit_1 | Fruit_2 | Fruit_3 |
|    0    |    0    |    1    |  → Apple (001)
|    0    |    1    |    0    |  → Banana (010)
|    0    |    1    |    1    |  → Cherry (011)
|    1    |    0    |    0    |  → Date (100)
|    1    |    0    |    1    |  → Elderberry (101)
```

✅ When to use:

- When the number of categories is **large** (high-cardinality features like ZIP codes, product IDs).
- Helps avoid curse of dimensionality.

🧩 Common Scenario: Encoding a "Country" column in a global dataset

Let's say we're working with a **"Country"** column in a customer data table, with **10,000+ unique countries or territories** (say it's a scraped ecommerce dataset including regions, zones, etc.).

CustomerID	Country	Purchase_Amount
1	India	500
2	USA	1200
3	Tuvalu	200
4	Brazil	800
...	...	...

🧪 How Each Encoding Handles It:

🟨 One-Hot Encoding

- **Creates 10,000+ new columns.**
- Too sparse, **huge memory footprint.**
- Might **slow down training** drastically.

🟦 Dummy Encoding

- Slightly better — creates 9,999 columns.
- Still **not practical.**

🟩 Binary Encoding

- **Only ~14 columns** (`log2(10000) ≈ 14`)
- **Compact, efficient** representation.
- Minimal loss of information.

✅ **Winner: Binary Encoding** for high-cardinality categorical data.

🔚 Summary Comparison Table

Encoding Type	Pros	Cons	Best Use Case
One-Hot Encoding	Simple, intuitive, preserves category info	High dimensionality with many categories	Small number of categories
Dummy Encoding	Avoids multicollinearity, compact vs one-hot	Still high dimensionality for many categories	Regression models (Linear, Logistic)
Binary Encoding	Compact, scalable for high-cardinality	Not easily interpretable, some info compression	High-cardinality columns (city, zip, etc.)

Would you like me to add a small code snippet for each in Python (Pandas/Scikit-learn style)?

Please come up with a better scenario / example

Certainly! Let's explore **Binary Encoding**, **One-Hot Encoding**, and **Dummy Encoding** with a practical scenario that highlights how each method handles the data differently and which might be preferable.

Scenario: **Predicting Housing Prices with 'Neighborhood' as a Categorical Feature**

Imagine you're developing a model to predict housing prices in a city. One of the features is the **'Neighborhood'** where each house is located. Suppose there are **8 unique neighborhoods** in your dataset:

- **Neighborhoods:** `['Downtown', 'Uptown', 'SuburbA', 'SuburbB', 'SuburbC', 'SuburbD', 'SuburbE', 'SuburbF']`

---

1. One-Hot Encoding

- **Method:** Creates a binary column for each neighborhood. Each row has a '1' in the column corresponding to its neighborhood and '0's elsewhere.
- **Columns Created:** Equal to the number of unique categories (8 in this case).

Example:

Neighborhood	Downtown	Uptown	SuburbA	SuburbB	SuburbC	SuburbD	Su
Downtown	1	0	0	0	0	0	0
SuburbB	0	0	0	1	0	0	0

Considerations:

- **Pros:**
  - Simple and intuitive.
  - Preserves the uniqueness of each category without implying any ordinal relationship.

- **Cons:**
  - Adds **8 additional columns**, increasing the dataset's dimensionality.
  - For models like **linear regression**, including all dummy variables can lead to **multicollinearity**, where one variable can be predicted from others, causing issues in model estimation.

---

2. Dummy Encoding

- **Method:** Similar to one-hot encoding but drops one category to serve as a **baseline**. This avoids multicollinearity by ensuring that the encoded variables are independent.
- **Columns Created:** Number of unique categories minus one (7 in this case).

Example:

Neighborhood	Uptown	SuburbA	SuburbB	SuburbC	SuburbD	SuburbE	Subu
Downtown	0	0	0	0	0	0	0
SuburbB	0	0	1	0	0	0	0

Interpretation:

- **Baseline Category:** 'Downtown' is the baseline. A row with all zeros corresponds to 'Downtown'.
- **Pros:**
  - Reduces the number of columns, mitigating multicollinearity.
  - Suitable for models sensitive to multicollinearity, like linear regression.
- **Cons:**
  - The choice of the baseline category can influence the interpretation of the model coefficients.

---

3. Binary Encoding

- **Method:** Converts each category into a unique binary code and then splits these binary digits into separate columns.
- **Columns Created:** `log2(number of categories)`, rounded up. For 8 categories, `log2(8) = 3`, so 3 columns.

Encoding Steps:

1. Assign a unique integer to each neighborhood:
    - Downtown: 0
    - Uptown: 1
    - SuburbA: 2
    - SuburbB: 3
    - SuburbC: 4
    - SuburbD: 5
    - SuburbE: 6
    - SuburbF: 7
2. Convert these integers to binary:
    - Downtown: `000`
    - Uptown: `001`
    - SuburbA: `010`
    - SuburbB: `011`
    - SuburbC: `100`
    - SuburbD: `101`
    - SuburbE: `110`
    - SuburbF: `111`
3. Split the binary digits into separate columns:

Neighborhood	Binary_1	Binary_2	Binary_3
Downtown	0	0	0
Uptown	0	0	1

Neighborhood	Binary_1	Binary_2	Binary_3
SuburbA	0	1	0
SuburbB	0	1	1

Considerations:

- **Pros:**
    - **Efficient**: Reduces the number of new columns to 3, regardless of the number of categories.
    - **Scalable**: Particularly beneficial for features with **high cardinality** (many unique categories).
- **Cons:**
    - The encoded columns are less interpretable.
    - Introduces a form of ordinal relationship, which may not be appropriate for all categorical data.

Comparison and Recommendation

Scenario Context:

- **Number of Categories:** 8 neighborhoods.
- **Model Type:** Linear regression (sensitive to multicollinearity).

Encoding Method Suitability:

- **One-Hot Encoding:**
    - **Pros:** Preserves categorical uniqueness.
    - **Cons:** Adds 8 columns; potential multicollinearity issues.
    - **Use Case:** Suitable for models not sensitive to multicollinearity or when interpretability is crucial.
- **Dummy Encoding:**
    - **Pros:** Reduces to 7 columns; avoids multicollinearity.
    - **Cons:** Requires careful selection of the baseline category.

    - **Use Case:** Preferred for linear regression models to prevent multicollinearity.
- **Binary Encoding:**
    - **Pros:** Reduces to 3 columns; efficient for high-cardinality data.
    - **Cons:** Less interpretable; may introduce unintended ordinal relationships.
    - **Use Case:** Best for features with a large number of categories.

Recommendation:

For predicting housing prices with 8 neighborhoods using a **linear regression model**, **Dummy Encoding** is advisable. It balances dimensionality reduction and mitigates multicollinearity while maintaining interpretability. However, if the number of neighborhoods were significantly higher (e.g., 50+), **Binary Encoding** could be considered for its efficiency, keeping in mind the potential trade-offs in interpretability.

Note: The choice of encoding technique should align with the specific dataset characteristics and the requirements of the predictive model being employed.

Sources

Great. And what is target encoding?

Target Encoding is a technique in machine learning used to convert categorical variables into numerical values by replacing each category with a statistic (typically the mean) of the target variable for that category. This method is particularly useful for handling high-cardinality categorical features. Codefinity

---

🔍 How Target Encoding Works

For each category in a categorical feature, target encoding calculates the average of the target variable for all data points belonging to that category. This average is then used to replace the categorical value. Machine Learning Interview

Example:

Consider a dataset with a categorical feature `City` and a target variable `Salary`:

Analytics Vidhya

City	Salary
Mumbai	100k
Delhi	80k
Mumbai	120k
Bangalore	90k
Delhi	85k

Calculating the average salary for each city: Machine Learning Interview

- Mumbai: (100k + 120k) / 2 = 110k
- Delhi: (80k + 85k) / 2 = 82.5k
- Bangalore: 90k

After target encoding, the `City` feature is transformed: Machine Learning Interview +2

City	Salary
110k	100k
82.5k	80k
110k	120k
90k	90k
82.5k	85k

✅ Advantages of Target Encoding

- **Efficient for High Cardinality:** Reduces dimensionality compared to one-hot encoding, which can create a large number of features for high-cardinality categorical variables.

- **Captures Target Relationship:** Incorporates the relationship between the categorical feature and the target variable, potentially improving model performance. Train in Data Blog

---

⚠️ Challenges and Mitigation Strategies

- **Overfitting:** Categories with few observations may lead to overfitting. Codefinity
  - *Mitigation:* Apply smoothing techniques that combine the category mean with the overall mean, weighted by the number of observations. Codefinity +1
- **Data Leakage:** Using target information from the entire dataset can leak information from the validation or test sets into the training process. Codefinity
  - *Mitigation:* Use cross-validation techniques to compute the target encoding within each fold, ensuring that the encoding for validation data is based only on training data.

📌 When to Use Target Encoding

- **High-Cardinality Categorical Features:** When a categorical feature has many unique categories, target encoding can be more practical than one-hot encoding.
- **Models Sensitive to Feature Dimensionality:** Algorithms like gradient boosting machines or neural networks can benefit from the reduced dimensionality.

In summary, target encoding is a powerful technique for transforming categorical variables, especially those with high cardinality, by leveraging the relationship between the feature and the target variable. However, it requires careful implementation to avoid overfitting and data leakage.

Sources