Preprocessing Tools

Cracking the Code: A Guide to Encoding Categorical Data for ML

Prepared by:
Hiba Mahdi
Sara Ali
Zeenah Shamil
Noor-AlHuda Ayad

Supervised by: Assist. Prof. Dr. Suhad F. Shihan

Table of Content:

1. Introduction

Preprocessing is a fundamental step in computer science that prepares data or program text for subsequent analysis, modeling, or compilation. In machine learning and artificial intelligence applications, preprocessing is essential for building high-quality datasets for model training. It includes activities such as data cleaning, integration, normalization, and reduction, which ensure that input data is suitable for analysis and model development. Optimizing preprocessing significantly boosts the performance of analytics, including predictive accuracy, although no universal pipeline fits every dataset and algorithm[1]

1.1 Preprocessing in Machine Learning

Preprocessing is a critical stage in machine learning, encompassing data cleaning, feature scaling, encoding, dimensionality reduction, and feature engineering. Data cleaning addresses inconsistencies, noise, and missing values, which can arise from equipment malfunction, communication interruptions, or data removal due to inconsistency. Missing values are handled through strategies such as deletion, mean imputation, regression imputation, least squares regression, local least squares, and K-nearest neighbor imputation. Imputation aims to replace missing data with estimated values, while deletion is appropriate only for missing completely at random (MCAR) cases; otherwise, it may introduce bias. Duplicate data is managed by identifying and removing redundant entries using knowledge-based methods or extraction, transformation, and loading (ETL) techniques. Feature scaling and normalization are essential for algorithms sensitive to feature magnitude, such as K-nearest neighbor and gradient boosting. Standard scaling (Z-score normalization) transforms features to have a mean of 0 and standard deviation of 1, ensuring balanced input and helping to speed up the convergence of gradient descent optimization methods. One-hot encoding is widely used to convert categorical variables into binary vectors, facilitating compatibility with mathematical models. This process increases dataset dimensionality and computational complexity but removes hierarchical order that could mislead models. Dimensionality reduction techniques, such as principal component analysis (PCA), project data into lower-dimensional spaces while retaining most variance, reducing computational effort and overfitting. PCA is applied after normalization and is commonly used in domains like network traffic analysis. Feature extraction and selection methods, including genetic algorithms, filter out irrelevant or redundant features, improving model efficiency and predictive performance. Wrapper, filter, and embedded methods are employed for feature selection, with wrapper methods evaluating subsets based on classifier performance and filter methods relying on statistical criteria.

In healthcare, cleaning and dimensionality reduction improved breast cancer classification accuracy from 54% to 67%. In Internet of Things (IoT) applications, normalization, filtering, and feature selection techniques are used to handle noisy sensor data and enhance model robustness. Feature selection eliminates redundant or irrelevant features, reduces overfitting, and increases interpretability. Scalability, noisy data, and domain-specific requirements present ongoing challenges in preprocessing.

2. Encoding

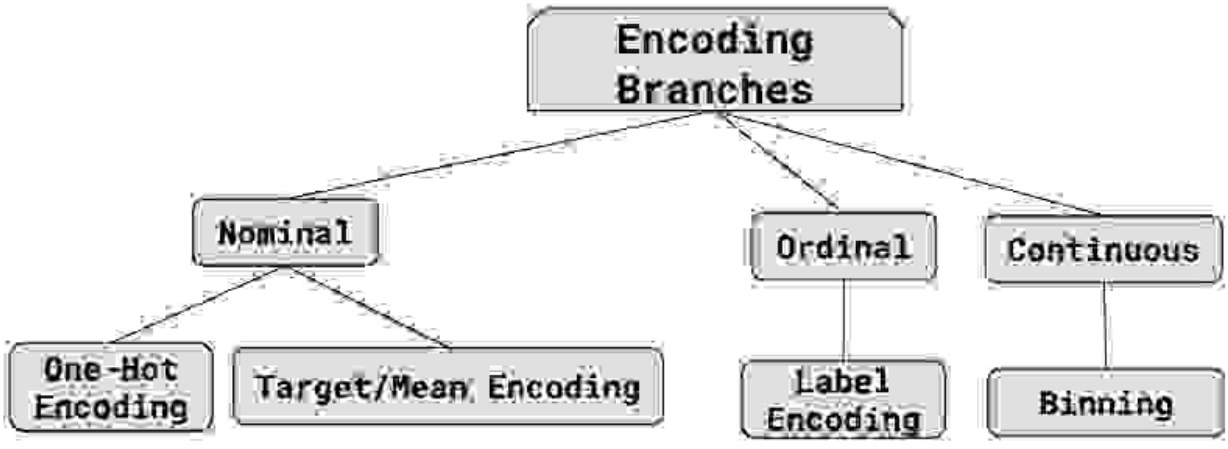
Data Encoding is a critical process in data management and machine learning that involves converting data from one format or structure to another to ensure compatibility and efficiency in processing. In the context of computing and data science, encoding is primarily used to transform raw data into a format that can be easily and effectively processed by algorithms, especially in cases where the initial data format is unsuitable for direct use in computations or applications.
Data encoding is the process of converting information into a specific format for efficient storage and transmission. Mastering data encoding is essential for students interested in fields like computer science, software development, and data analysis, as it underpins how digital information is structured and accessed [2]. This is crucial because many algorithms work best with numerical input, so encoding converts data like text or categories into a machine-readable form, enabling pattern identification, storage, transmission, and efficient processing.

2.1 Machines Speak Math, Humans Speak Categories

We may ask “Can we feed any text directly into a math equation (eg. m*x + b)?” The answer is absolutely no! Machine learning models are mathematical. They require numerical input. Our job is to convert these categories into numbers, but we have to do it smartly. And this is accomplished by using Encoding.

2.2 Encoding Branches

Encoding has three categories that are Nominal, Ordinal, Continuous each with its own subbranches that we will discuss four of in this lecture.



2.2.1 Nominal Encoding

Nominal features are those where the categories have no inherent order or ranking, such as colors, types of animals. Nominal data consists of categories that have no inherent order. Examples include colors

(red, blue, green), types of animals (cat, dog, bird), or brands (Nike, Adidas, Puma). These categories are merely labels and cannot be logically ordered or ranked.

Machine learning algorithms typically require numerical input, so categorical data must be converted into a numerical format. Proper encoding ensures that the model can interpret and learn from the data effectively. Let's dive into some popular encoding techniques [3].

2.2.1.1 One-Hot Encoding

One-hot encoding is one of the most widely used methods for encoding nominal data. It creates a new binary column for each category in the original feature. Each row will have a 1 in the column corresponding to its category and 0s in all other columns. The presence of a category is marked with a 1, and its absence with a 0 [3].

Original Data
Red
Blue
Green
Red

One-Hot Encoded		
Color_Blue	Color_Green	Color_Red
0	0	1
1	0	0
0	1	0
0	0	1

It represents each category equally without implying any order or relationship. The model can learn independently for each color.

```
#importing pandas library
import pandas as pd
# Let us create a sample dictionary with categories
data = {'Color':['Red', 'Blue', 'Green', 'Red']}
#create data frame for the original data
df = pd.DataFrame(data)
# Perform One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=['Color'])
print("\n One-Hot Encoded Data:\n", df_encoded)
```

```
 Output: One-Hot Encoded Data:
    Color_Blue  Color_Green  Color_Red
0       False        False       True
1        True        False      False
2       False         True      False
3       False        False       True
```

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. It has many functions that we can use with Encoding branches that we will discuss some of them in this lecture.

DataFrame() is one of pandas functions that is two-dimensional, size-mutable, potentially heterogeneous tabular data. Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

get_dummies() is a function that converts categorical variables into dummy/indicator variables. Each variable is converted into as many 0/1 variables as there are different values. Columns in the output are each named after a value; if the input is a DataFrame, the name of the original variable is prepended to the value.

Pros:

1. Prevents the model from incorrectly assuming an ordinal relationship between categories.
2. Simple and easy to implement.
3. can handle multiple categories in an unbiased manner towards any particular category [7].

Cons:

1. Can lead to a high-dimensional and sparse dataset if there are many unique categories, which can slow down training .
2. It can also create sparsity in the dataset, making machine learning algorithms difficult to identify patterns [7].

2.2.1.2 Target/Mean Encoding

A target encoding is any kind of Nominal Encoding that replaces a feature's categories with some number derived from the target. A simple and effective version is to apply a group aggregation, like the mean [4]. It is preferred to be used with high-cardinality features (e.g., postal codes, product IDs) where One-Hot would create too many columns.

```
import numpy as np
postalTarget_data = {
    'postal_code': ['90210', '10001', '30301', '90210', '10001'],
    'house_price': [2500000, 1500000, 800000, 2300000, 1600000]}
df = pd.DataFrame(postalTarget_data)
#Calculate the mean of the target variable ('postal_code') for each category
target_mapping = df.groupby('postal_code')['house_price'].mean()
# Map the mean values back to the original DataFrame
df['postalCode_encoded'] = df['postal_code'].map(target_mapping)
print(df)
```

```
Output: postal_code  house_price  postalCode_encoded
0         90210      2500000           2400000.0
1         10001      1500000           1550000.0
2         30301       800000            775000.0
3         90210      2300000           2400000.0
4         10001      1600000           1550000.0
```

```
5        30301       750000             775000.0
6        90210      2400000            2400000.0
7        10001      1550000            1550000.0
```

NumPy stands for Numerical Python is an open source Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important [5].
mean() is one of NumPy's functions that computes the mean of groups, excluding missing values.

groupby() is one of pandas functions Group DataFrame using a mapper or by a Series of columns. A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.
map() is a pandas function that is applied to a Dataframe elementwise. This method applies a function that accepts and returns a scalar to every element of a DataFrame.

Target Encoding is great for high-cardinality features: A feature with a large number of categories can be troublesome to encode: a one-hot encoding would generate too many features and alternatives, like a label encoding, might not be appropriate for that feature. A target encoding derives numbers for the categories using the feature's most important property: its relationship with the target. It is also great for domain-motivated features: From prior experience, you might suspect that a categorical feature should be important even if it scored poorly with a feature metric. A target encoding can help reveal a feature's true informativeness [4].
Pros:

- It does not increase the dimensionality of the dataset and can capture meaningful relationships between the categorical feature and the target.

Cons:

- Requires careful implementation, often with cross-validation or smoothing, to prevent data leakage and overfitting.

2.2.2 Ordinal Encoding

Ordinal encoding consists of converting categorical data into numeric data by assigning a unique integer to each category, and is a common data preprocessing step in most data science projects. It is particularly useful when an inherent ordering or ranking is present within the categorical variable. For example, the variable size with values small, medium, and large exhibits a clear ranking, i.e., small < medium < large, thereby making ordinal encoding an appropriate encoding method [6].

2.2.2.1 Label Encoding

Label encoding is another method for transforming categorical data into numerical data. Label encoding assigns each category a unique integer value which means it converts categorical text data into

numerical values by assigning a unique integer to each category [7]. For example, suppose we have a categorical variable “Size” with three categories (Small, Medium, and Large). In that case, the label encoding process would assign integer values 0 to Small, 1 to Medium 2 to Large.

```
# create and print dictionary as a DataFrame
data = {'Size': ['Small', 'Medium', 'Large']}
df = pd.DataFrame(data)
# Encode a column
df['Size_Encoded'] = pd.factorize(df['Size'])[0]
print("DataFrame with Size encoded:\n", df)
```

```
Output: DataFrame with Size encoded:
      Size  Size_Encoded
0    Small             0
1   Medium             1
2    Large             2
```

factorize() is a pandas function that encodes the object as an enumerated type or categorical variable. This method is useful for obtaining a numeric representation of an array when all that matters is identifying distinct values.

Pros:

1. Simple and easy to implement.
2. Does not increase dimensionality.

Cons:

1. Introduces an ordinal relationship that may not exist and can create order bias, where the integer values assigned to each category can influence the machine learning algorithm’s output.
2. It can also create difficulties when comparing categories, as the integer values do not necessarily represent the actual relationship between the categories [3, 7].

2.2.3 Continuous Encoding

Continuous encoding is the branch of encoding that deals with the form of fractional numbers. It can be the version of an android phone, the height of a person, the length of an object, etc. Continuous data represents information that can be divided into smaller levels. The continuous variable can take any value within a range [8].

2.2.3.1 Binning (Discretization)

Discretization, also known as binning, is the process of transforming continuous numerical data into discrete intervals or bins. This is particularly useful when dealing with numerical data that has a wide range of values or outliers. By grouping these values into bins, we can simplify the data and make it more manageable for machine learning models [9]. For example, consider a dataset containing the ages of individuals ranging from 0 to 100. Instead of treating age as a continuous variable, we can group ages into intervals like 0–12, 12–18, 18–65, and so on. This not only reduces the complexity of the data but also helps in handling outliers.

```
# Sample age data
ages = [2, 5, 12, 18, 25, 30, 38, 45, 50, 62, 70]
df = pd.DataFrame({'Age': ages})
# Binning with custom ranges and labels
bins = [0, 12, 18, 65, np.inf] # Define bin edges
labels = ['Child', 'Teenager', 'Adult', 'Senior'] # Define labels for
each bin
df['Age_Group_Custom'] = pd.cut(df['Age'], bins=bins, labels=labels,
right=False)
print("Binning with custom ranges:\n", df)
```

```
Output: Binning with custom ranges:
     Age   Age_Group_Custom
0     2             Child
1     5             Child
2    12          Teenager
3    18             Adult
4    25             Adult
5    30             Adult
6    38             Adult
7    45             Adult
8    50             Adult
9    62             Adult
10   70            Senior
```

cut() is a pandas function (Bin values into discrete intervals) it is used when you need to segment and sort data values into bins. This function is also useful for going from a continuous variable to a categorical variable. For example, cut could convert ages to groups of age ranges. Supports binning into an equal number of bins, or a pre-specified array of bins.

Pros:

- It makes model results easier to understand by transforming complex numerical data into clear, human-readable categories (e.g., ages grouped into "Adult" or "Senior" bins)

Cons:

- Can lead to information loss by treating all values within a bin identically.

2.3 Choosing Appropriate Model

It might be confusing by now to decide which model to choose so for a quick overview you must know that choosing the right encoding method or any preprocessing method depends on **the nature of your data**, the **machine learning model you plan to use**, and the **cardinality (number of unique values) of the categorical features**.

- Target encoding: When you have high-cardinality features (a large number of unique values) with many unique categories in your dataset.

- One-hot encoding: When you have nominal (unordered) categorical features with low cardinality (a small number of unique values).
- Label encoding: When You have ordinal categorical features with a clear, meaningful order (e.g., "low," "medium," "high").
- Binning: When you want to simplify a numerical feature and reduce its complexity.

A direct link for the practical code.

References

1. Preprocessing - an overview | ScienceDirect Topics
2. Data Encoding: Techniques & Examples | StudySmarter
3. Data Science: Guide to Encoding Nominal Categorical Features. | by Ajaykumar Dev | Medium
4. Target Encoding
5. Introduction to NumPy
6. Ordinal Encoding — 1.8.3
7. Data Encoding a normalization techniques in machine learning
8. 4 Types of Data - Nominal, Ordinal, Discrete, Continuous
9. Encoding Numerical Data with Discretization and Binning | by Sneh Paghdal | Medium.