Lecture: Working with Categorical Data in Machine Learning

1. Introduction and Learning Objectives

In this lecture, we will examine how categorical data is represented and used in machine learning models. By the end of this session, you should be able to:

- Distinguish categorical data from numerical data
- Explain why categorical data must be encoded numerically
- Apply one-hot encoding and vocabulary encoding
- Identify and handle common issues such as outliers and high dimensionality
- Understand alternatives such as embeddings, hashing, and feature crosses

This lecture assumes familiarity with basic machine learning concepts and numerical feature handling.

---

2. What Is Categorical Data?

Categorical data consists of values drawn from a **finite or countable set of distinct categories**, rather than continuous numerical quantities.

Examples of Categorical Data

- Species of animals in a national park
- Street names in a city
- Email classification: spam vs. not spam
- Exterior house colors
- Binned numerical values (for example, age groups)

Key Distinction: Categorical vs. Numerical

Numerical data supports meaningful arithmetic operations. For example:

- A house with 200 square meters is roughly twice the size of a house with 100 square meters.

Categorical data does **not** support such relationships. Treating categorical identifiers as numbers can introduce misleading assumptions.

---

3. Numbers Can Be Categorical

Not all numbers should be treated as numerical features.

Example: Postal Codes

- Postal codes are integers, but their numeric magnitude has no semantic meaning.
- Treating postal code `20004` as “twice” `10002` is incorrect.
- Postal codes should therefore be modeled as **categorical**, not numerical, features.

Principle:
If numeric values do not have an intrinsic order or scale that matters, they should be treated as categorical.

---

4. Why Encoding Is Necessary

Machine learning models can only train on **floating-point numerical values**. They cannot operate directly on strings such as `"Red"` or `"Dog"`.

Encoding is the process of converting categorical values into numerical representations that models can learn from.

---

5. Vocabulary Encoding and Feature Dimensionality

Feature Dimension

The **dimension** of a feature refers to the number of elements in its vector representation.

Some categorical features are **low-dimensional**, meaning they have a small, fixed number of categories:

Feature	Number of Categories
snowed_today	2
skill_level	3
season	4
day_of_week	7
planet	8

For low-dimensional features, vocabulary-based encoding is appropriate.

---

6. Indexing Categorical Values

The first step in vocabulary encoding is mapping each category to a **unique integer index**.

Example: `car_color`

Color	Index
Red	0
Orange	1
Blue	2
Yellow	3
Green	4
Black	5
Purple	6
Brown	7

However, indexed integers **cannot** be fed directly into a model. Doing so would cause the model to interpret these indices as continuous values.

---

7. One-Hot Encoding

Definition

One-hot encoding converts each category into a binary vector where:

- The vector length equals the number of categories
- Exactly one element is `1.0`
- All other elements are `0.0`

Example: `car_color` (8 categories)

Color	One-Hot Vector
Red	[1, 0, 0, 0, 0, 0, 0, 0]
Blue	[0, 0, 1, 0, 0, 0, 0, 0]
Black	[0, 0, 0, 0, 0, 1, 0, 0]

The model learns **one weight per category**, allowing it to treat each category independently.

Note: Multi-Hot Encoding

In multi-hot encoding, multiple positions can be `1.0`. This is used when multiple categories apply simultaneously.

---

8. Sparse Representation

Most one-hot vectors contain mostly zeros and are therefore **sparse**.

Sparse Representation Concept

Instead of storing the full vector, we store only the index (or indices) of non-zero values.

- One-hot vector for `"Blue"`:
  `[0, 0, 1, 0, 0, 0, 0, 0]`

- Sparse representation:
  `2`

This reduces memory usage significantly. Internally, however, models still operate on the expanded vector form.

---

9. Outliers in Categorical Data

Categorical features can include **rare or unusual categories**.

Example

`car_color` may include rare values like `"Mauve"` or `"Avocado"`.

Solution: Out-of-Vocabulary (OOV) Bucket

- Rare categories are grouped into a single **OOV category**
- The model learns one shared weight for all rare values
- This improves robustness and reduces noise

---

10. High-Dimensional Categorical Features

Some categorical features have **very large vocabularies**:

Feature	Approx. Categories
English words	~500,000
US postal codes	~42,000
German last names	~850,000

Why One-Hot Encoding Fails Here

- Extremely large feature vectors

- High memory consumption
- Slower training and inference

---

11. Alternatives to One-Hot Encoding

11.1 Embeddings (Preferred)

- Map categories into low-dimensional dense vectors
- Capture semantic similarity between categories
- Improve training speed and inference latency

(Embeddings are covered in a separate module.)

11.2 Feature Hashing (Hashing Trick)

Hashing maps categories into a fixed number of bins.

Steps:

1. Choose number of bins `N`
2. Apply a hash function to each category
3. Compute `hash(category) % N`
4. One-hot encode based on resulting bin index

Trade-off:

- Reduced dimensionality
- Possible hash collisions

---

12. Common Pitfalls and Best Practices

- Never train directly on raw strings
- Do not treat categorical IDs as numerical values
- Use OOV buckets for rare categories
- Avoid one-hot encoding for very large vocabularies
- Prefer embeddings for high-cardinality features

---

13. Knowledge Check

True or False:
A machine learning model can train directly on raw string values like `"Red"` and `"Black"`.

Answer: False
Models require floating-point numerical inputs; categorical values must be encoded first.

---

14. Summary

In this lecture, we covered:

- The nature of categorical data
- Why encoding is required
- Vocabulary and one-hot encoding
- Sparse representations and OOV handling
- Challenges of high-dimensional categorical features
- Alternatives such as embeddings and hashing

Understanding how to correctly represent categorical data is essential for building accurate and efficient machine learning models.

Lecture: Categorical Data – Feature Crosses

1. Introduction

In this lecture, we introduce **feature crosses**, a technique used to help machine learning models learn **interactions** between categorical features. Feature crosses are especially useful for **linear models**, which otherwise struggle to capture nonlinear relationships.

---

2. What Is a Feature Cross?

A **feature cross** is created by taking the **Cartesian product** of two or more categorical (or bucketed numerical) features.

- Each crossed feature represents a **specific combination** of values
- The model learns a separate weight for each combination
- Feature crosses allow linear models to model **nonlinear patterns**

---

3. Why Feature Crosses Matter

Linear models assume features act independently. However, in many real-world problems, **feature interactions matter**.

Feature crosses:

- Encode interactions between features
- Allow linear models to learn conditional relationships
- Are conceptually similar to polynomial features for numerical data

---

4. Example: Leaf Classification

Consider a dataset used to classify tree species based on leaf characteristics.

Base Features

1. **edges**
    - smooth
    - toothed
    - lobed
2. **arrangement**
    - opposite
    - alternate

Each feature is one-hot encoded.

---

5. One-Hot Encoded Representation

Assume the following order:

- edges = (smooth, toothed, lobed)
- arrangement = (opposite, alternate)

A leaf with **smooth edges** and **opposite arrangement** is represented as:

- edges = (1, 0, 0)
- arrangement = (1, 0)

---

6. Creating the Feature Cross

The feature cross of `edges × arrangement` produces all possible combinations:

- Smooth_Opposite
- Smooth_Alternate

- Toothed_Opposite
- Toothed_Alternate
- Lobed_Opposite
- Lobed_Alternate

Each crossed feature is computed by **multiplying** the corresponding one-hot values.

---

7. Feature Cross Calculation

Each crossed feature is defined as:

- Smooth_Opposite = edges[0] × arrangement[0]
- Smooth_Alternate = edges[0] × arrangement[1]
- Toothed_Opposite = edges[1] × arrangement[0]
- Toothed_Alternate = edges[1] × arrangement[1]
- Lobed_Opposite = edges[2] × arrangement[0]
- Lobed_Alternate = edges[2] × arrangement[1]

---

8. Example Output

If a leaf has **lobed edges** and **alternate arrangement**, the crossed feature vector becomes:

```
{0, 0, 0, 0, 0, 1}
```

Only **Lobed_Alternate** is active.

This representation allows the model to learn that this specific combination may strongly indicate a particular tree species.

---

9. Feature Crosses vs. Polynomial Features

Feature crosses are similar in spirit to polynomial transforms:

Polynomial Features	Feature Crosses
Combine numerical features	Combine categorical features
Capture nonlinear effects	Capture feature interactions
Used with continuous data	Used with discrete data

---

10. When to Use Feature Crosses

Feature crosses are most effective when:

- Domain knowledge suggests meaningful interactions
- Individual features alone are insufficient
- Linear or shallow models are being used

Examples:

- Location × time of day
- User type × device type
- Leaf edge × leaf arrangement

---

11. Caution: Sparsity Explosion

Feature crosses can dramatically increase feature space size.

Example:

- Feature A: 100 categories
- Feature B: 200 categories
- Crossed feature size: **20,000 categories**

This results in:

- Increased sparsity
- Higher memory usage
- Slower training

Feature crosses should be applied **selectively**.

---

12. Alternatives

When feature interactions are complex or unknown:

- Neural networks can automatically learn feature combinations
- Embeddings can capture interactions in a dense representation

---

13. Summary

- Feature crosses combine categorical features to model interactions
- They help linear models learn nonlinear patterns
- They are powerful but can significantly increase sparsity
- Use domain knowledge to decide which features to cross

Feature crosses are a practical and interpretable way to enhance categorical feature representations when used carefully.