databricks

Data Preparation and Feature Engineering

LECTURE

Data Encoding

In this lecture, we cover data encoding, focusing on handling categorical features, using label encoding for ordinal data, and understanding embeddings and how they work.

Data Encoding

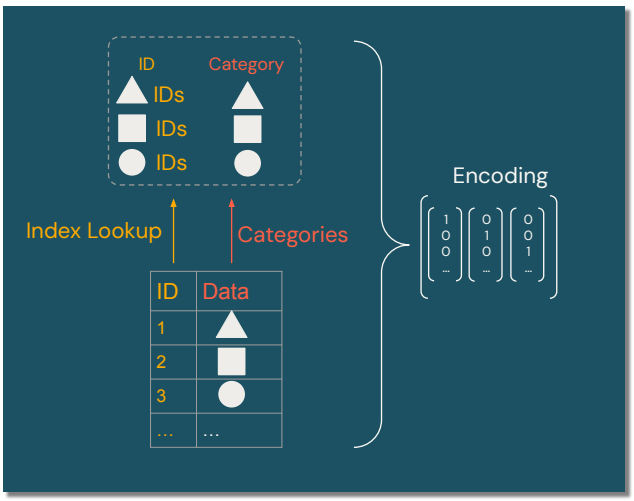
Why Encoding?

Data encoding is a important pre-processing step in **preparing categorical data for machine learning algorithms**, as vast majority of algorithms accept numerical input exclusively.

Issues with classic ML

- Handling high-cardinality features
- Introducing unintended relationships
- Overfitting
- Increased computational cost
- Possible lack of interpretability

Encoding Process



Encoding is the process of transforming categorical or non-numeric data into numerical representations that machine learning models can process. This is necessary because models require floating-point values for training.

- Each encoding method comes with trade-offs, including considerations for memory usage, interpretability, and the risk of overfitting.
- Preprocessing should account for new or unseen categories in test data to ensure robust model performance.
- The choice of encoding depends on the model type—what is effective for tree-based models may not be suitable for linear models.

Working with Categorical Features

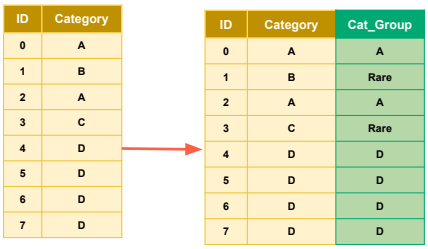
High Cardinality

Issue: Large set of categories

Many unique values in a categorical feature can lead to a large number of dummy variables, increasing dimensionality and potentially causing issues. This is known as the high-cardinality problem.

Possible Solutions:

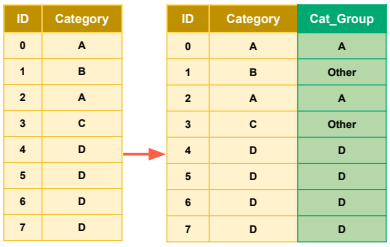
Group Rare Categories:



ID	Category
0	A
1	B
2	A
3	C
4	D
5	D
6	D
7	D

ID	Category	Cat_Group
0	A	A
1	B	Rare
2	A	A
3	C	Rare
4	D	D
5	D	D
6	D	D
7	D	D

Top-N Categories:



ID	Category
0	A
1	B
2	A
3	C
4	D
5	D
6	D
7	D

ID	Category	Cat_Group
0	A	A
1	B	Other
2	A	A
3	C	Other
4	D	D
5	D	D
6	D	D
7	D	D

Group Rare Categories: Group infrequent categories into an "Other" category.

Top-N Categories: Keep only the most frequent categories and group the rest

Working with Categorical Features

Missing Values

Issue: Categorical gaps

Categorical features often have missing values, which need to be addressed before model training.

Possible Solutions:

Imputation:

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	NaN
2	NaN	30.0
3	4.0	40.0
4	5.0	50.0

→

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	32.5
2	3.0	30.0
3	4.0	40.0
4	5.0	50.0

Consider Missing as a Separate Category:

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	NaN
2	NaN	30.0
3	4.0	40.0
4	5.0	50.0

→

ID	Feature A	Feature B
0	1.0	10.0
1	2.0	-1.0
2	-1.0	30.0
3	4.0	40.0
4	5.0	50.0

Imputation: Use methods like mode imputation for nominal variables or create a separate category for missing values.
Consider Missing as a Separate Category: Treat missing values as a distinct category.

Working with Categorical Features

Encoding Categories

Issue: String types

Models require numerical input, and categorical variables need to be encoded.

Possible Solutions:

One-Hot Encoding

ID	Category
0	A
1	B
2	A
3	C
4	A

→

ID	Cat_A	Cat_B	Cat_C
0	1	0	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0

Label Encoding:

ID	Category
0	A
1	B
2	A
3	C
4	A

→

ID	Category	Label
0	A	0
1	B	1
2	A	0
3	C	2
4	A	0

Ordinal Encoding:

ID	Category
0	A
1	B
2	A
3	C
4	A

→

ID	Category	Ordinal
0	A	0.0
1	B	1.0
2	A	0.0
3	C	2.0
4	A	0.0

One-Hot Encoding: Convert each category into a binary column.
Label Encoding: Assign unique numerical values to each category.
Ordinal Encoding: Assign numerical values based on the ordinal relationship of categories.

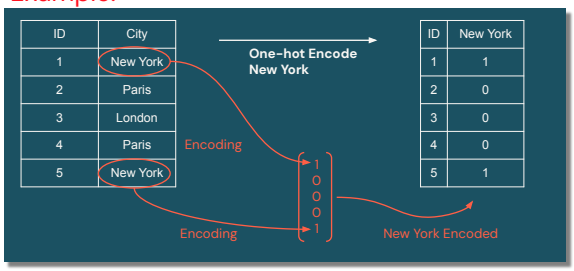
One-hot Encoding

Represent categorical variables as binary vectors

Procedure:

1. **Add binary columns** (0 or 1) for each unique value in categorical columns.
2. Each category is now a column that map to either 0 or 1 in reference to the ID column

Example:



ID	City
1	New York
2	Paris
3	London
4	Paris
5	New York

ID	New York	Paris	London
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0
5	1	0	0

Drawbacks of One-Hot Encoding:

- **Sparsity:** Creates many binary columns, increasing memory use and inefficiency.
- **Tree Model Limits:** Excessive splits can hurt decision tree performance.
- **High Cardinality Issues:** Inefficient with variables having many unique values.
  Obscured Importance: Binary expansion makes feature importance harder to interpret.
- **Higher Costs:** Larger feature space raises training time and resource demands.

Takeaway: Use selectively, and consider alternatives for high-cardinality or large datasets.

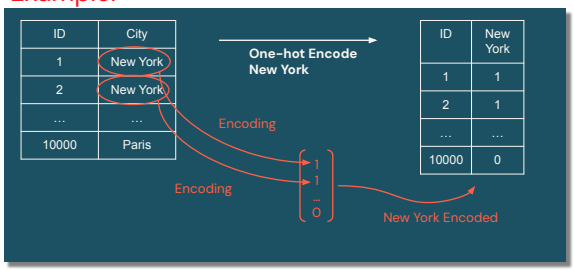
One-hot Encoding

Represent categorical variables as binary vectors

Drawbacks:

1. Induces sparsity
2. Limites split options for tree-based models
3. Inefficiency with high-cardinality variables
4. Obscures feature importance
5. Increases computational costs

Example:



ID	City
1	New York
2	Paris
3	London
4	Paris
5	New York

ID	New York	Paris	London
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0
5	1	0	0

Drawbacks of One-Hot Encoding

- **Sparsity:** Generates many binary columns, leading to high memory use and inefficiency.
- **Tree Model Limits:** Excessive splits may hurt performance in decision trees and ensembles.
- **High Cardinality Issues:** Becomes inefficient with variables having many unique values.
- **Obscured Importance:** Too many columns make feature importance harder to interpret.
- **Higher Costs:** Expanded feature space raises training time and resource needs.

Takeaway: Use one-hot encoding selectively; consider alternatives for high-cardinality or large datasets.

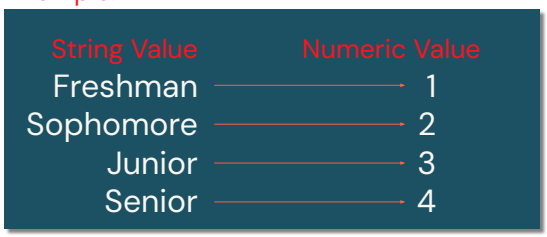
Label Encoding for Ordinal Features

Convert categorical data into numerical labels *(aka "String Indexing")*

Procedure:

1. **Assign numeric labels:** Map each category to a numeric value based on its natural order.
2. **Transform the Feature:** Replace each categorical value in the feature column with its corresponding numeric label.

Example:



ID	High School Grade Level	Age
1	Freshman	14
2	Senior	17
3	Junior	16
4	Freshman	15
5	Sophomore	16

ID	High School Grade Level	Age
1	1	14
2	4	17
3	3	16
4	1	15
5	2	16

Drawbacks of Label Encoding for Ordinal Features:

- **Arbitrary Values:** Numeric labels may not reflect actual differences (e.g., "Low" = 1, "High" = 2).
- **Misinterpreted Relationships:** Implies linear gaps that don't match real ordinal distances.
- **Model Sensitivity:** Algorithms may pick up unintended bias from assigned values.
- **Information Loss:** Discards the true magnitude of differences between categories.
- **Limited Compatibility:** Distance-based models (e.g., clustering) may misinterpret encoded values.

  Takeaway: Choose encoding methods carefully to preserve ordinal meaning and model compatibility.

Label Encoding for Ordinal Features

Convert categorical data into numerical labels *(aka "String Indexing")*

Drawbacks:

1. Arbitrary Assignments Can Mislead Models.
2. Misinterpretation of Relationships.
3. Model Sensitivity.
4. Loss of Information.
5. Not Suitable for all algorithms.

Example:

String Value		Numeric Value
Freshman	Less Important	1
Sophomore		2
Junior	More Important	3
Senior		4

ID	High School Grade Level	Age
1	Freshman	14
2	Senior	17
3	Junior	16
4	Freshman	15
5	Sophomore	16

ID	High School Grade Level	Age
1	1	14
2	4	17
3	3	16
4	1	15
5	2	16

Drawbacks of Label Encoding for Ordinal Features:

- **Arbitrary Values:** Numeric labels don't reflect true category differences (e.g., "Low" = 1, "High" = 2).
- **False Relationships:** Implies linear gaps that may misrepresent ordinal meaning.\
- **Model Sensitivity:** Algorithms can pick up unintended bias from numeric magnitude.
- **Information Loss:** Ignores real distance between categories, losing nuance.
- **Algorithm Limits:** Distance-based models (e.g., clustering) may misinterpret encoded values.

Takeaway: Use carefully—encoding must align with data characteristics and algorithm requirements.

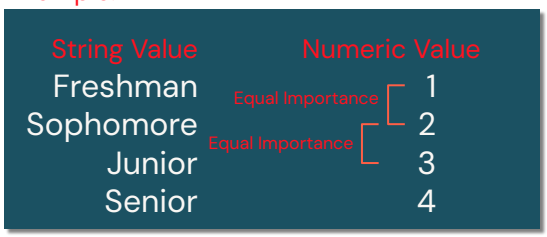
Label Encoding for Ordinal Features

Convert categorical data into numerical labels *(aka "String Indexing")*

Drawbacks:

1. Arbitrary Assignments Can Mislead Models.
2. Misinterpretation of Relationships.
3. Model Sensitivity.
4. Loss of Information.
5. Not Suitable for all algorithms.

Example:



ID	High School Grade Level	Age
1	Freshman	14
2	Senior	17
3	Junior	16
4	Freshman	15
5	Sophomore	16

ID	High School Grade Level	Age
1	1	14
2	4	17
3	3	16
4	1	15
5	2	16

Drawbacks of Label Encoding for Ordinal Features:

- **Arbitrary Values:** Numeric labels are arbitrary (e.g., "Low" = 1, "High" = 2) and may not reflect reality.
- **Misleading Relationships:** Implies linear gaps between categories that don't match true ordinal differences.
- **Model Sensitivity:** Algorithms can misinterpret magnitude, introducing bias.
- **Information Loss:** Discards real differences between categories.
- **Algorithm Issues:** Distance-based models may misread encoded values, harming performance.

Takeaway: Pick encoding methods that align with both data properties and algorithm needs.

Target Encoding

Encoding categorical variables: replacement by mean of target variable

Procedure:

1. **Compute the mean** of the target variable for each category.
2. **Substitute** category instances with their mean values.

ID	City	Clicked
1	New York	1
2	Paris	0
3	London	1
4	Paris	1
5	New York	0

ID	City	Clicked
1	0.5	1
2	0.5	0
3	1.0	1
4	0.5	1
5	0.5	0

Example:

City	Clicked Values	Mean
New York	[1,0]	0.5
Paris	[0,1]	0.5
London	[1]	1

Drawbacks of Target Encoding:

- **Overfitting Risk:** Sensitive to training data; requires smoothing or regularization for generalization.
- **New Categories:** Unseen categories (e.g., new city) need fallback handling.
- **Target Dependence:** Skewed or imbalanced targets can distort encodings.
- **Small Dataset Limits:** Requires enough data for stable estimates; small samples reduce reliability.
- **Bias Risk:** Rare categories may create biased or noisy encodings.

Takeaway: Use target encoding carefully—apply safeguards to ensure fairness, robustness, and generalization

Target Encoding

Encoding categorical variables: replacement by mean of target variable

Drawbacks:

1. Overfitting risk
2. Handling new categories
3. Dependency of target distribution
4. Limited applicability to small datasets
5. Risk of bias

Example:

City	Clicked Values	Mean
New York	[1,0]	0.5
Paris	[0,1]	0.5
London	[1]	1

Can lead to overconfident prediction

ID	City	Clicked
1	New York	1
2	Paris	0
3	London	1
4	Paris	1
5	New York	0

ID	City	Clicked
1	0.5	1
2	0.5	0
3	1.0	1
4	0.5	1
5	0.5	0

Drawbacks of Target Encoding:

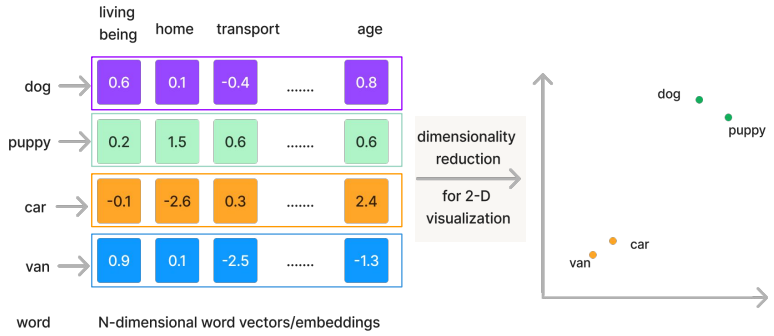
- **Overfitting:** Can fit too closely to training data; smoothing or regularization is needed.
- **New Categories:** Unseen values (e.g., new city) require fallback strategies.
- **Target Dependence:** Skewed or imbalanced targets distort encoding quality.
- **Small Datasets:** Limited data produces unreliable estimates.
- **Bias Risk:** Rare categories may introduce bias, hurting predictions.

Takeaway: Apply target encoding with care, using safeguards to maintain robustness and fairness.

What are Embeddings?

Embeddings: Representing Data in a Meaningful Way

- Embeddings are ***dense vector representations*** of data, capturing semantic relationships.
- Unlike traditional encoding methods, embeddings ***place similar items closer together*** in a lower-dimensional space.
- Used widely in ***text***, ***categorical data***, ***images***, and ***audio processing***.



Embeddings are dense vector representations of data that help capture relationships between different items.

Unlike traditional encoding methods like one-hot encoding, embeddings map items into a lower-dimensional space where similar items are placed closer together.

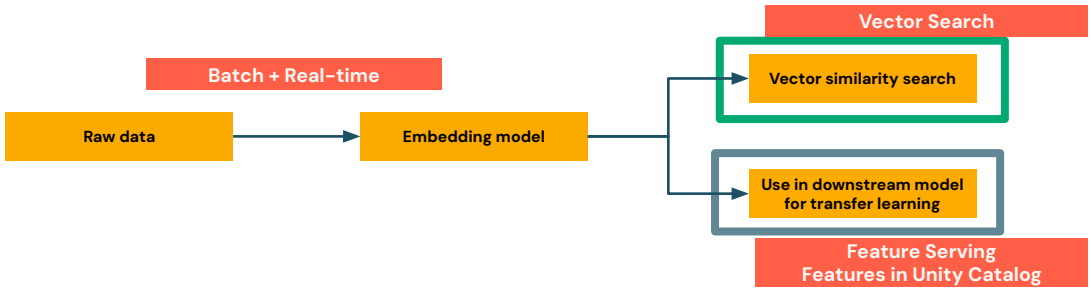
They are used in various domains, including text, categorical variables, images, and audio.

The example on the left shows embeddings for different words like "dog", "puppy", "car", and "van" represented as vectors. On the right, dimensionality reduction is applied to visualize these vectors in 2D, where we can see that similar words are positioned near each other.

How Embeddings Work?

The Role of Embeddings in Machine Learning

- Embeddings transform high-dimensional categorical or textual data into a **compact**, **dense vector space**.
- These representations capture relationships and context among different entities.
- Used in **Recommendation Systems**, **NLP**, **Image Search**, **and more**.
- Can be learned from data using neural networks or retrieved from **pretrained models** (e.g., Word2Vec, FastText).



Embeddings convert raw categorical or textual data into a compact, dense vector space. These vectors capture relationships between data points.
They can be used in various applications like recommendation systems, NLP, and image search.
Embeddings can be generated using models trained on the dataset or retrieved from pretrained models like Word2Vec or FastText.

The diagram shows raw data being processed through an embedding model. The output vectors can be used for similarity search or passed to another model for tasks like transfer learning.

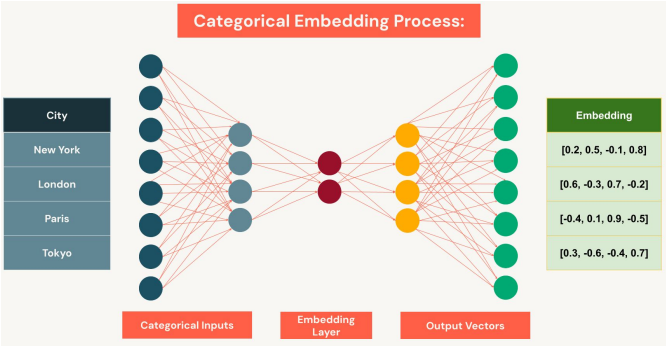
Categorical Embedding

Transforming categorical variables into meaningful, continuous vectors

- Categorical embeddings replace sparse categorical encoding methods with lower-dimensional, dense vector representations.
- These embeddings capture relationships between different categories based on learned patterns.

Process:

1. Convert categorical variables into unique indices.
2. Pass through an embedding layer (neural network or learned lookup table).
3. The model assigns dense vector representations to each category based on learned relationships.



Categorical embeddings are used to represent categorical variables as dense vectors.

Instead of using one-hot or label encoding, this method maps categories into lower-dimensional continuous vectors.

The process starts by converting categories into numeric indices. These are then passed through an embedding layer, which can be a neural network or a learned lookup table.

The model outputs vector representations for each category, capturing relationships between them based on patterns learned during training.

The diagram shows how inputs like city names are passed through an embedding layer to produce output vectors.

Embeddings vs. One-Hot Encoding

Choosing the Right Encoding for Categorical Data

Feature	One-Hot Encoding	Embeddings
Representation	Sparse binary vectors (high-dimensional)	Dense numerical vectors (low-dimensional)
Semantic Relationships	Does not capture relationships between categories	Places similar categories closer in vector space
Scalability	Inefficient for high-cardinality data	Scales well with large category sets
Efficiency	Inefficient for high-cardinality features (sparse matrix)	Efficient for high-cardinality features
Model Suitability	Suitable for simple models like Decision Trees	Best for Neural Networks and deep learning models
Example:	TV → [1, 0, 0, 0] Laptop → [0, 1, 0, 0] Phone → [0, 0, 1, 0]	TV → [0.6, 1.2, -0.8] Laptop → [0.5, 1.1, -0.7] Phone → [0.2, -0.4, 1.5]

Let's walk through the differences between one-hot encoding and embeddings for handling categorical data.

One-hot encoding creates a binary vector for each category. It's easy to apply but leads to high-dimensional and sparse data, especially with many unique categories.

Embeddings convert categories into dense, low-dimensional vectors. These are learned during training and can group similar categories closer in vector space.

In terms of scalability and efficiency, embeddings are more suitable for high-cardinality features. One-hot encoding can become inefficient and memory-heavy in such cases.

For models, one-hot encoding works well with simpler algorithms like decision trees. Embeddings are commonly used with neural networks and deep learning approaches.

At the bottom, we can see the difference in representation:

- One-hot uses binary vectors like `[1, 0, 0, 0]` for TV.
- Embeddings show continuous vectors like `[0.6, 1.2, -0.8]` for the same category.

Thank you for completing this lesson and continuing your journey to develop your skills with us.