NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Feature engineering: Data encoding, scaling and Bias mitigation methods

Smitha K G
Senior Lecturer
College of Computing and Data Science

Data encoding

- Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the models to give and improve the predictions.
- Categorical data encoding is a fundamental step in preparing data for machine learning models, as most algorithms require numerical input.
- There can be two kinds of categorical data:
  - Nominal data (No intrinsic order data)
  - Ordinal data
- The choice of encoding technique depends heavily on the nature of the categorical variable (nominal or ordinal)

Types of encoding

- Label Encoding (Ordinal)
- One-Hot Encoding (nominal)
- Count / Frequency encoding (nominal)
- Binary Encoding (nominal)
- Target Encoding (nominal)
- Hash Encoding (nominal)

Label encoding

- This type of encoding is used when the variables in the data are ordinal. Ordinal encoding converts each label into unique integer values based on its rank.
- Only really useful if there exist a natural order in categories
  - Model will consider one category to be 'higher' or 'closer' to another
- Pros: Simple, memory-efficient, and effective for preserving the order of features.
- Cons: If the categories are not ordinal, the model will mistakenly assume an order and a linear relationship, which is often incorrect.

One Hot encoding

- In One-Hot Encoding (OHE)- for nominal data-each category of any categorical variable creates a new binary column for each unique category. After OHE, the number of one hot variables depends on the number of categories in the data
- Pros:
  - Creates independent features, prevents the model from assuming an artificial order (prevents misinterpretation), and is suitable for most linear (logistic regression, SVM) and distance-based models (K nearest Neighbors).
  - Provides clarity/transparency
  - OHE allows the model to learn the presence of each category, providing the most expressive and non-biased representation of nominal data
- Cons:
  - Curse of Dimensionality if a variable has many unique categories (high cardinality), increasing memory usage and computation time.
  - It also causes multicollinearity (as one column can be perfectly predicted by the others, known as the dummy variable trap).

Count/Frequency encoding

- This type of encoding is used when the data is nominal.
- Count encoding technique replaces each category with the frequency (or count) of its occurrence in the dataset.
- Pros:
  - Does not increase the dimensionality of the dataset (dimensionality reduction).
  - Captures information about the prevalence of each category.(rare vs frequent)
  - Handles new or rare Categories: Rare or infrequent categories are assigned a small, distinct value, grouping them naturally and reducing their potential to cause overfitting. Unseen categories in the test set can be assigned a default frequency (e.g., 0) found in the training data.
- Cons: If two different categories have the same frequency, the model will treat them identically, leading to a loss of information.

Comparison for nominal

Feature	One-Hot Encoding (OHE)	Frequency Encoding
Cardinality	Best for **Low** Cardinality (<5 categories).	Best for **High** Cardinality (>10 categories).
Output Dimension	**Increases** dimensionality (1 column per category).	**Maintains** dimensionality (1 numerical column total).
Risk of Issue	Risk of **High Dimensionality** and **Sparsity**.	Risk of **Collision** (two categories get the same frequency).
Model Type Fit	**Linear** (LogReg, SVM), **NN**, and Distance-based.	**Tree-based** (Random Forest, XGBoost, CatBoost).

Binary encoding

- Useful when cardinality is high as one hot encoding will create a greater number of columns
- This saves vast amounts of memory and speeds up model training when dealing with high-cardinality data.
- Binary encoding doesn’t assume linear relationship between categories hence mitigating the risk of misleading linear models
- Minimum information loss: Compared to Frequency Encoding (which can merge information for categories with the same count), Binary Encoding preserves the uniqueness of every category without collision.
- Cons
  - Loss of Interpretability: The resulting binary features (columns) are abstract and do not directly relate to the original category.
  - Artificial Relationships: While it doesn't assume a simple linear order, the binary features still introduce a fixed numerical relationship between categories that may not exist. The model must learn that certain combinations of binary features correspond to distinct categories.
  - Not suitable for simple linear models

Hash encoding

- Feature Hashing, or the Hashing Trick, is a method used primarily for encoding nominal categories, especially those with very high cardinality (especially in NLP)
- Like One-Hot encoding, the hash encoder converts the category into binary numbers using new data variables but here we can fix the number of new data variables (no dramatical increase in dimension).

Target encoding

- Target Encoding is for converting high cardinality nominal categorical features into numerical values.
- The core idea is to replace each category with the average value of the target variable observed for that category. Enc(i) = Mean(Target |Category = i)
  - If the target is binary, the encoded value is the proportion of observations in that category belonging to the positive class.
  - If the target is continuous (like price), the encoded value is the average value of the target for that category.

Encoding Type	Reason
Ordinal / Label	Natural order (Small < Medium < Large)
One-Hot	No order, low cardinality
Frequency Encoding	Great for high-cardinality & risk for collision.
Binary Encoding	Medium cardinality → reduces dimensions
Hash Encoding	scalable for larger vocabularies
Target Encoding	Strongly correlated with another feature → ideal use case

Data/Feature scaling

Changes the magnitude and range of the data but preserves the shape of the underlying distribution.

Use when different numeric features have different scales (different range of values)

Features with much higher values may overpower the others

Goal: bring them all within the same range

Key point: Interpretability

Mechanism: Applies a linear transformation (subtraction/division) to compress or expand the data based on its mean, standard deviation, min, or max.

Why it's used: To ensure all features contribute equally to the distance calculation in distance-based algorithms (KNN, SVM) and to speed up the convergence of gradient-based optimizers.

Why we need scaling (linear)

In a linear model, the equation for prediction is:

$$y = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b$$

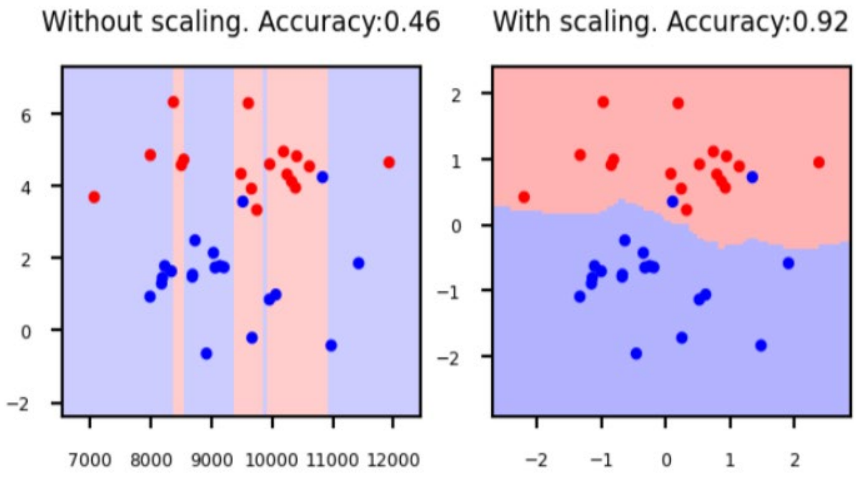
where w_i are the **weights** (or coefficients) and x_i are the **features**.

- If Feature x_1 ranges from 0 to 100,000 (like income) and Feature x_2 ranges from 0 to 1 (like a binary flag), the model's weight for x_1 (w_1) must be very small to keep its overall contribution comparable to x_2 's contribution. Conversely, w_2 will be much larger. For example, w_1 =0.001 and w_2 =5. You cannot simply compare w_1 and w_2 to determine which feature is more "important" because they are defined on different input scales.
- **With Scaling (Standardization/Normalization):**
  - all features are on a comparable scale (e.g., all centered around 0 with a standard deviation of 1).
  - Consequently, the resulting learned weights (w'1, w’2, …) will also have similar scales.
  - if w'1 is larger than w'2, it **directly implies** that Feature 1 is a stronger predictor of the target variable than Feature 2.

Why we need scaling

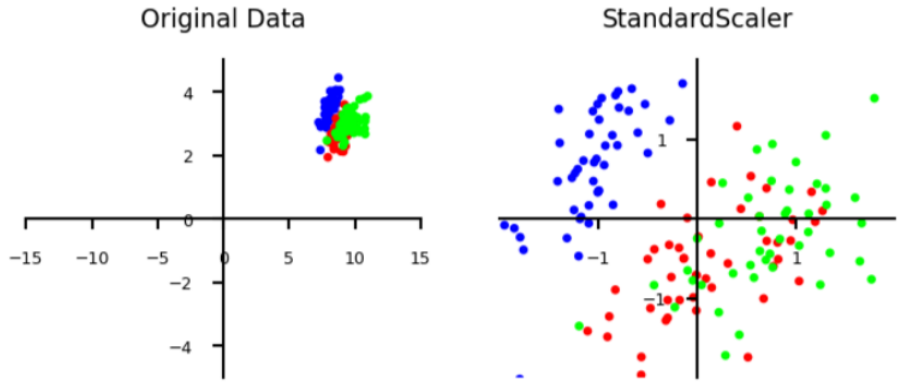
- Distance based algorithms (KNN, SVM, K means)-Ensures all features contribute equally to the distance calculation, preventing large-magnitude features from dominating.

- Feature Magnitude Dominance: In unscaled data, features with a large magnitude dominate the distance calculation ignoring the contribution of features with smaller magnitudes , leading to a biased and incorrect classification.
- Equal Contribution and Accuracy: Scaling transforms all features to a comparable range (e.g., [0, 1]), ensuring they contribute equally to the distance calculation.



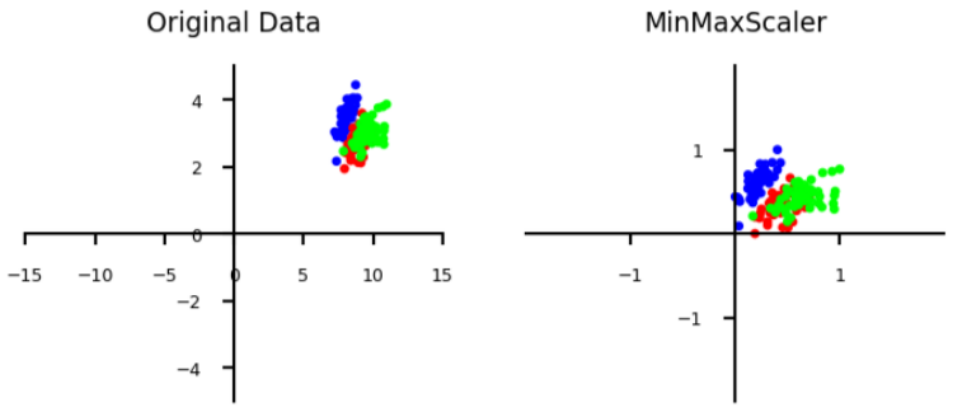
Standardization

- Standardization (Z-score) assumes data is more or less normally distributed
- New feature has μ=0 and σ=1, values can still be arbitrarily large $x_{new}=(x-\mu)/\sigma$
- Centered at 0, unit variance (no fixed bounds)
- Best for algorithms assuming a **Gaussian distribution** (e.g., PCA) and is less affected by **outliers**.



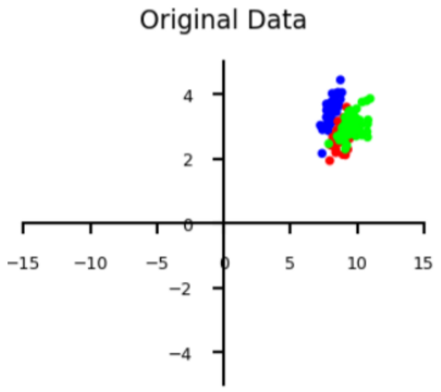
Normalization (Min-Max)

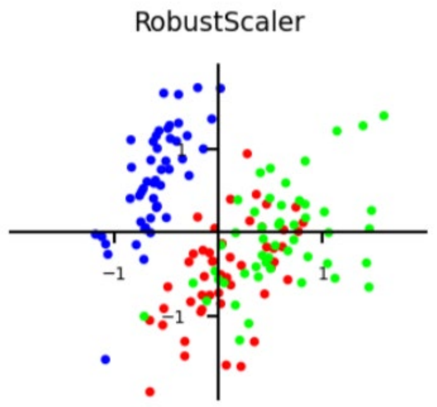
- New feature x_{new}=(x–min)/(max-min)
- Fixed range, typically [0, 1] or [-1, 1]
- Best for algorithms that require bounded input values (e.g., certain **Neural Networks**) but is highly sensitive to **outliers**.



Robust Scaling

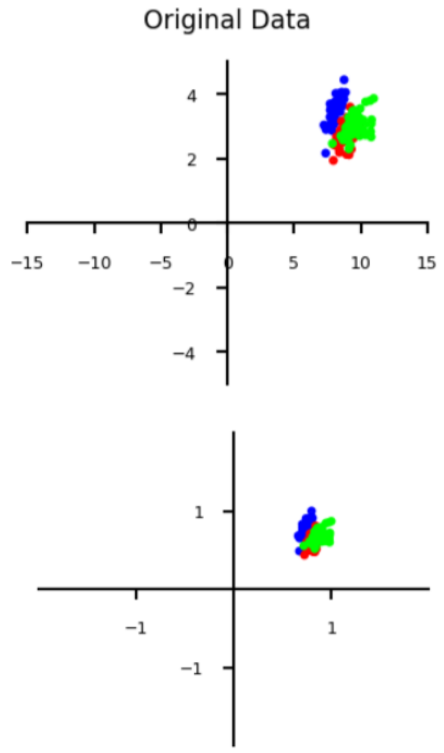
- Feature scaling method designed to be **resistant to outliers**.
- x_{new}=(x−median(x))/IQR) (Subtracts the median, scales between quantiles q25 and q75)
- Recommended scaler when your dataset contains **significant outliers** that you do not want to remove but also do not want to corrupt your feature scaling
- Similar to standard scaler, but ignores outliers





Maximum Absoluter Scalar

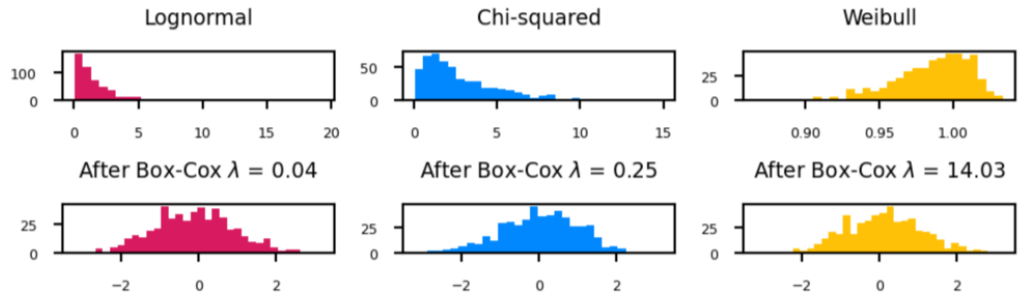
- MaxAbsScaler: Rescales each feature individually based on its **maximum absolute value**.
- For sparse data (many features, but few are non-zero)
  - Maintain sparseness (efficient storage)
- Scales all values so that maximum absolute value is 1
- Similar to Min-Max scaling without changing 0 values
- Does Zero Preservation: scaling does not involve centering the data
- Outlier sensitive



Power transformation

- Some features follow certain distributions
  - E.g. number of twitter followers is log-normal distributed
- Box-Cox transformations transform these to normal distributions (λ is fitted)
  - Only works for positive values, use Yeo-Johnson otherwise

$$bc_\lambda(x) = \begin{cases} log(x) & \lambda = 0 \\ \frac{x^\lambda - 1}{\lambda} & \lambda \neq 0 \end{cases}$$



Advanced Feature Engineering

Feature Creation (Derived Features):

Example: Creating Age from Date of Birth or Price/SqFt from Price and Area.

Textual Feature Extraction (NLP):

TF-IDF: Measures word importance based on frequency in a document vs. corpus.

Word2Vec: Creates dense vectors that capture semantic word relationships.

Dimensionality Reduction:

PCA (Principal Component Analysis): Reduces complexity while retaining variance.

Impact of Bias on Fairness

Bias directly influences whether the model treats different groups equitably.

Historical Bias: Data reflects past inequalities. E**x:** Past hiring records favor men → hiring model predicts men as better candidates.

Sampling Bias: Some groups are underrepresented. **Ex:** Face recognition dataset mostly contains light-skinned faces → model performs poorly on darker-skinned individuals.

Label Bias: The labels themselves are unfair or reflect human judgment errors. Ex: Loan default labels based on biased approval systems.

Measurement Bias: Features are measured differently across groups. Ex**:** Different medical devices work better on one demographic than another.

Impact of Bias on Model Generalization

Overfitting to Dominant Groups
When data overrepresents one group, the model *specializes* in it and performs poorly elsewhere.
Example: A medical diagnosis model trained mainly on adults → fails on children.
Poor Feature Learning
Biased data may push the model to learn shortcuts.
Example: A model predicts "wolf vs dog" based on background snow instead of the animal → fails on new images.
Domain Shift Problems
If training data is not diverse, the model cannot adapt to new environments.
Example: Self-driving car trained only in sunny conditions → fails in rain or night-time.
Unrealistic Assumptions (Algorithmic Bias)
Simplified linear assumptions may generalize poorly when reality is more complex.
Example: Using linear models for non-linear relationships.

How to Reduce Bias, Improve Fairness, Generalization

- Improve dataset diversity
- Balanced sampling and reweighting
- Fair labeling practices
- Use fairness-aware algorithms
- Cross-group evaluation metrics (FPR, FNR, demographic parity)
- Regular auditing and bias detection tools
- Human-in-the-loop review for high-risk decisions

Bias mitigation techniques

- **Pre-processing Techniques** (Intervene on the Data):These methods modify the training data before it's fed into the model to reduce or eliminate bias.
- **Reweighing (or Re-sampling):** Assigning a higher weight to samples from an underrepresented (unprivileged) group that received an unfavorable outcome, and a lower weight to overrepresented samples.
- **Oversampling :**Oversampling is a method used to balance the dataset when the protected group (unprivileged group) or the unfavorable outcome (e.g., loan rejection) is significantly underrepresented.

Conclusion & Key Takeaways

Feature Engineering is both an art & science requiring domain knowledge and experimentation.

Better Features → Better Models: Effective transformation leads to simpler, more powerful models.

Actionable Advice: Choose scaling/encoding methods based on algorithm and data distribution.