Module 3: Feature Engineering

Overview

Feature engineering is the process of transforming raw data into representations that machine learning models can use effectively. This module covers how to handle multiple features, scale data appropriately, encode categorical variables, create interaction features, and select the most relevant features. Good feature engineering often matters more than the choice of algorithm.

---

1. Multivariate Linear Regression

Extending to Multiple Features

With one feature: `y = w x + b`

With multiple features: `y = w x + w x + ... + w x + b`

Matrix Notation

Matrix notation makes multivariate regression elegant:

```
y = X @ w + b
```

Where: - **X**: Matrix of shape (n_samples, n_features) - **w**: Weight vector of shape (n_features,) - **b**: Bias (scalar) - **y**: Predictions of shape (n_samples,)

Gradient Computation

The gradient computation extends naturally:

```
L/ w = (2/n) × X  @ (predictions - targets)
```

The matrix transpose X distributes the error across all features appropriately.

---

2. Feature Scaling

Why Scale Features?

Features on different scales cause problems: - Gradient descent converges slowly (elongated loss surface) - Features with large values dominate the model - Regularization penalizes features unequally

Example problem: - Feature 1: Age (0-100) - Feature 2: Income (0-1,000,000)

Without scaling, income will dominate.

Min-Max Normalization

Scales features to a fixed range, typically [0, 1]:

```
x_scaled = (x - x_min) / (x_max - x_min)
```

When to use: - When you need bounded values - When the distribution is roughly uniform - For neural networks with bounded activations

Z-Score Standardization

Centers data to mean=0 and scales to standard deviation=1:

```
x_scaled = (x -  ) /
```

When to use: - When features should be comparable - When algorithms assume normally distributed data - Most common default choice

Log Transformation

Compresses data with a long tail:

```
x_scaled = log(1 + x)
```

When to use: - For highly skewed distributions - When data spans orders of magnitude (e.g., income, populations) - When multiplicative relationships matter

---

3. Handling Missing Data

Types of Missingness

- **MCAR** (Missing Completely at Random): No pattern
- **MAR** (Missing at Random): Related to observed data
- **MNAR** (Missing Not at Random): Related to unobserved data

Strategies

Deletion: - Remove rows or columns with missing values - Simple but loses data - Risky if missingness is informative

Imputation with Statistics: - Fill with mean (common, sensitive to outliers) - Fill with median (robust to outliers) - Fill with mode (for categorical data)

Imputation with Models: - Predict missing values from other features - More sophisticated but complex

Indicator Variables: - Add binary column indicating missingness - Preserves information that data was missing

---

4. Encoding Categorical Variables

The Problem

Machine learning models need numbers, but categorical data is symbolic (e.g., “red”, “blue”, “green”).

One-Hot Encoding

Create a binary column for each category:

Color	red	blue	green
red	1	0	0
blue	0	1	0
green	0	0	1

Pros: - No implied ordering - Works for any categorical variable

Cons: - High cardinality □ many columns - Sparse representation

Label Encoding

Assign integer to each category: - red □ 0 - blue □ 1
- green □ 2

Warning: This implies an ordering (green > blue > red) that may not exist. Only use for ordinal categories like size (small < medium < large).

Target Encoding (Advanced)

Replace category with the mean target value for that category. Powerful but risks overfitting—requires careful cross-validation.

---

5. Feature Crosses

Capturing Interactions

Sometimes the effect of one feature depends on another. Feature crosses capture these interactions.

Example: Effect of apartment size on price depends on city: - 500 sq ft in NYC = expensive - 500 sq ft in rural Kansas = cheap

Creating Crosses

For categorical × categorical: Concatenate values: (city=NYC, size=small) □ “NYC_small”

For numerical × numerical: Multiply: $x \times x$

For categorical × numerical: Create separate features for each category

When to Use

- When you suspect interactions
- When a linear model otherwise underfits
- When domain knowledge suggests it

---

6. Bucketing (Binning)

Converting Continuous to Discrete

Sometimes a continuous feature has a non-linear relationship with the target that's better captured as buckets.

Example: Age might have different effects in ranges: - 0-18: Child - 18-35: Young adult - 35-55: Middle age - 55+: Senior

Implementation

```
# Define bin edges
bins = [0, 18, 35, 55, 100]

# Assign to buckets
bucket = np.digitize(age, bins)
```

When to Use

- Non-linear relationships that are stepwise
- When exact values don't matter
- When combined with one-hot encoding

---

7. Feature Selection

Why Select Features?

- **Reduce overfitting**: Fewer features = less opportunity to memorize
- **Improve interpretability**: Know what matters
- **Speed up training**: Less computation
- **Remove noise**: Irrelevant features hurt performance

Filter Methods

Evaluate features independently of the model: - Correlation with target - Mutual information - Variance threshold

Pro: Fast, model-agnostic **Con:** Ignores feature interactions

Wrapper Methods

Evaluate subsets of features using the model: - Forward selection (add features one by one) - Backward elimination (remove features one by one) - Recursive feature elimination

Pro: Considers feature interactions **Con:** Computationally expensive

Embedded Methods

Feature selection built into model training: - L1 regularization (Lasso) drives weights to zero - Tree-based feature importance

Pro: Efficient, considers interactions **Con:** Model-specific

---

8. The Curse of Dimensionality

What It Is

As the number of features (dimensions) increases: - Data becomes sparse (points spread out) - Distances become meaningless (all points seem equidistant) - Exponentially more data is needed for coverage

Why It Matters

Volume concentration: In high dimensions, most of the volume of a hypercube is in the corners, far from the center. A hypersphere captures almost none of the space.

Distance meaninglessness: In high dimensions, the difference between the nearest and farthest neighbor approaches zero—everyone looks equally far away.

Mitigations

- Feature selection (reduce dimensions)
- Dimensionality reduction (PCA, covered in Module 8)
- Regularization (penalize complexity)
- More data (but need exponentially more)

Key Takeaways

1. **Multiple features**: Use matrix notation: y = X @ w + b
2. **Scaling is essential**: Z-score or min-max for comparable features
3. **Handle missing data thoughtfully**: Impute or use indicators
4. **Encode categoricals properly**: One-hot for nominal, label only for ordinal
5. **Feature crosses capture interactions**: Important for linear models
6. **Bucketing handles non-linearity**: Discretize for stepwise effects
7. **Select relevant features**: Reduce noise and overfitting
8. **Beware the curse of dimensionality**: More features require more data

---

Connections to Future Modules

- **Module 4**: Same preprocessing applies to logistic regression
- **Module 6**: Neural networks can learn feature crosses automatically
- **Module 8**: PCA for dimensionality reduction
- **Module 9**: Text features have special encoding (embeddings)

Further Reading

- [Links to be added]
- Google ML Crash Course: Feature Engineering
- Feature Engineering for Machine Learning (Alice Zheng)