Lecture: Numerical Data and Feature Vectors in Machine Learning

1. Introduction: How Models Actually Ingest Data

In many introductory explanations, machine learning models appear to operate directly on rows of a dataset. In practice, this is not how models consume data.

A dataset may contain many columns, but only a subset of those columns are used as **features**. Even then, the model does not ingest the raw dataset values directly. Instead, each training example is transformed into a **feature vector**.

---

2. Feature Vectors: The True Input to a Model

A **feature vector** is an ordered array of floating-point values representing a single example.

Key points:

- Each row in the dataset becomes one feature vector.
- Only selected features are included.
- Every value in the vector must be numeric (specifically floating-point).

Although it may seem intuitive to use raw dataset values, this approach often leads to poor model performance. Raw values can vary widely in scale, distribution, and meaning, making learning inefficient or unstable.

---

3. Feature Engineering: Why Transformation Is Necessary

Feature engineering is the process of transforming raw dataset values into representations that a model can learn from effectively.

Contrary to intuition, models typically learn better from *processed* values rather than raw values.

Common feature engineering techniques for numerical data include:

a. Normalization

- Converts numerical values into a standard range (for example, 0 to 1 or centered around 0).
- Prevents features with large numeric ranges from dominating the learning process.
- Improves numerical stability and convergence during training.

b. Binning (Bucketing)

- Converts continuous numerical values into discrete ranges or buckets.
- Useful when precise numeric differences are less important than relative ranges.
- Can help models capture non-linear relationships.

Feature engineering is not optional; it is a foundational step in building effective machine learning systems.

---

4. Requirement: All Features Must Be Numerical

Machine learning models operate on floating-point numbers. However, many real-world datasets contain:

- Strings
- Categories
- Boolean or structured values

A significant portion of feature engineering involves converting non-numerical data into numerical form. While this lecture focuses on numerical features, categorical data preprocessing is addressed separately in later modules.

---

5. First Steps When Working with Numerical Data

Before constructing feature vectors, it is essential to understand the numerical data itself. This should be done in two complementary ways:

---

6. Visualizing Numerical Data

Visualization helps reveal patterns, anomalies, and assumptions that may not be obvious from raw numbers.

Recommended practices:

- Use **scatter plots** to inspect relationships between features.
- Use **histograms** to understand distributions.
- Visualize data not only at ingestion, but throughout transformation steps.

Tools such as **pandas** are well-suited for these tasks, though the choice of visualization tool may depend on the data format (e.g., CSV vs. protocol buffers).

---

7. Statistical Evaluation of Numerical Features

Beyond visual inspection, numerical features should be evaluated statistically. Common metrics include:

- Mean and median
- Standard deviation
- Percentiles:
    - 0th percentile (minimum)
    - 25th percentile
    - 50th percentile (median)
    - 75th percentile
    - 100th percentile (maximum)

These statistics provide insight into scale, spread, and skewness.

---

8. Identifying and Handling Outliers

An **outlier** is a data point that lies far from most other values in a feature or label.

One heuristic for detecting outliers:

- If the difference between the 0th and 25th percentiles is very different from the difference between the 75th and 100th percentiles, outliers are likely present.

However, statistics alone are insufficient; visual inspection remains important.

---

9. Types of Outliers and How to Handle Them

Outliers generally fall into two categories:

a. Mistake Outliers

- Caused by data entry errors or instrument malfunctions.
- These examples should usually be removed.

b. Legitimate Outliers

- Represent real but rare phenomena.
- Decision criteria:
    - If the model must perform well on such cases, keep them.
    - If not, consider removing them or applying techniques such as **clipping**.

Extreme outliers, even when legitimate, can still negatively affect model training and must be handled carefully.

---

10. Key Takeaways

- Models ingest **feature vectors**, not raw dataset rows.
- Feature vectors contain only floating-point values.
- Feature engineering is critical for effective learning.
- Numerical data must be explored visually and statistically.
- Outliers require careful analysis, not automatic removal.

Understanding and preparing numerical data is a foundational skill in machine learning and directly impacts model performance and reliability.

ecture: Normalization of Numerical Data in Machine Learning

1. Motivation for Normalization

After examining numerical data using visualization and statistical techniques, the next step is to **transform the data** so that models can train more effectively. One of the most important transformations is **normalization**.

Normalization is the process of transforming numerical features so that they exist on a similar scale.

Consider two features:

- Feature X ranges from 154 to 24,917,482
- Feature Y ranges from 5 to 22

If these features are used together without normalization, the model will implicitly treat Feature X as far more important simply because its values are larger.

---

2. Why Normalization Matters

Normalization provides several critical benefits:

1. **Faster convergence during training**
   When feature ranges differ significantly, gradient descent can oscillate and converge slowly. Although adaptive optimizers such as Adam or Adagrad mitigate this issue, normalization still improves stability.

2. **Improved prediction quality**
   Models trained on normalized features often produce more reliable and generalizable predictions.

3. **Prevention of numerical instability (NaN trap)**
   Extremely large values can exceed floating-point precision limits, causing values to become NaN (“not a number”), which can corrupt the entire model.

4. **Balanced feature importance**
   Without normalization, models overweight features with large numeric ranges and

underweight those with smaller ranges, leading to suboptimal learned weights.

---

3. When Normalization Is Recommended

Normalization is strongly recommended when:

- Features span distinctly different ranges (e.g., age vs. income)
- A single feature spans a very wide range (e.g., city population)

Important rule:
If a feature is normalized during training, the *same normalization must be applied during inference*.

---

4. Consequences of Not Normalizing

Even modest differences in feature ranges can cause issues.

Example:

- Feature A ranges from –0.5 to +0.5
- Feature B ranges from –5.0 to +5.0

Although both ranges are small, Feature B spans ten times the range of Feature A. As a result:

- The model initially assumes Feature B is ten times more important
- Training takes longer
- The final model may be suboptimal

With extreme range disparities, training may fail entirely.

Normalization Techniques

This lecture covers four common techniques:

1. Linear scaling
2. Z-score scaling
3. Log scaling
4. Clipping

---

5. Linear Scaling (Min–Max Scaling)

Linear scaling converts values from their natural range into a standard range, typically **[0, 1]** or **[–1, +1]**.

Formula (0 to 1 scaling):
x′=x−xminxmax−xminx' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}x′=xmax−xminx−xmin

Where:

- xxx = original value
- xminx_{\min}xmin = minimum value in the dataset
- xmaxx_{\max}xmax = maximum value in the dataset

Example:

If a feature ranges from 100 to 900 and a value is 300:

x′=300−100900−100=0.25x' = \frac{300 - 100}{900 - 100} = 0.25x′=900−100300−100=0.25

When to Use Linear Scaling

Linear scaling works well when:

- The feature's bounds are stable over time
- The feature has few or no extreme outliers
- The distribution is approximately uniform

Example: Human age

- Natural bounds are known (roughly 0–100)

- Few extreme outliers
- Adequate coverage across the range

Limitation:
Most real-world features do not satisfy these conditions. As a result, linear scaling is often inferior to Z-score scaling.

---

6. Z-Score Scaling (Standardization)

A **Z-score** represents how many standard deviations a value is from the mean.

Formula:
z=x−μσz = \frac{x - \mu}{\sigma}z=σx−μ

Where:

- μ\muμ = mean
- σ\sigmaσ = standard deviation

Example:

- Mean = 100
- Standard deviation = 20
- Value = 130

z=130−10020=+1.5z = \frac{130 - 100}{20} = +1.5z=20130−100=+1.5

After Z-score scaling:

- Mean becomes 0
- Standard deviation becomes 1

---

7. Interpreting Z-Scores in Normal Distributions

In a classic normal distribution:

- 68.27% of values fall between –1 and +1
- 95.45% fall between –2 and +2
- 99.73% fall between –3 and +3

Values beyond ±4 are rare but not impossible, especially in very large datasets.

When to Use Z-Score Scaling

Z-score scaling is ideal when:

- The feature follows a normal or near-normal distribution
- The feature has a clear central tendency

Example: Adult height
Large populations produce a normal distribution with meaningful outliers that the model can learn from.

---

8. Combining Z-Score Scaling with Clipping

Some features are mostly normal but contain extreme outliers far from the mean (e.g., net worth).

In such cases:

- Apply Z-score scaling
- Clip extreme Z-scores (e.g., restrict values to –3 to +3)

This approach preserves the overall distribution while preventing extreme values from dominating training.

---

9. Log Scaling

Log scaling replaces a value with its logarithm, usually the natural logarithm.

Formula:
$x' = \ln(x)$

When to Use Log Scaling

Log scaling is effective for **power-law distributions**, where:

- Small values are common
- Large values are rare but extreme

Examples:

- Movie ratings
- Book sales
- Website traffic

Log scaling compresses large values, making differences more manageable for linear models.

Intuition Example:

- ln(100) ≈ 4.6
- ln(1,000,000) ≈ 13.8

Instead of being 10,000× larger, the scaled value is only about 3× larger.

---

10. Clipping

Clipping caps extreme values at a specified threshold.

Example:

- Feature: rooms per person
- Most values fall between 0 and 4
- A few extreme values reach as high as 17

By clipping values at 4:

- All values above 4 become 4
- Extreme influence is reduced

- The feature becomes more useful for training

Clipping does not remove data—it limits its impact.

---

11. When to Use Clipping

Clipping is appropriate when:

- Outliers are legitimate but too rare to learn from effectively
- Extreme values distort training

However:

- Some outliers carry important signal
- Clipping should be applied cautiously

Clipping is often combined with other normalization techniques.

---

12. Summary of Normalization Techniques

Technique	Best Used When Distribution Is
Linear scaling	Uniform, bounded, low outliers
Z-score scaling	Normal or near-normal
Log scaling	Heavy-tailed / power-law
Clipping	Contains extreme outliers

---

13. Key Takeaways

- Normalization is essential for effective training and stable convergence
- The “best” normalization depends on the feature’s distribution
- Visualization and statistics should guide normalization choices

- Normalization decisions directly affect model performance

Lecture: Binning (Bucketing) Numerical Data in Machine Learning

---

1. Introduction to Binning

Binning (also called **bucketing**) is a feature engineering technique that groups continuous numerical values into discrete ranges called *bins* or *buckets*.

In many cases, binning converts a **numerical feature into a categorical representation**, allowing models to learn patterns that are not well captured by linear relationships.

---

2. Why Binning Is Useful

Binning is particularly effective when:

- The relationship between a feature and the label is **nonlinear**
- Feature values are **clustered**
- Precise numeric differences are less important than **which range** a value falls into

Instead of forcing the model to learn a single linear weight, binning allows the model to learn **separate weights for different ranges** of the feature.

---

3. Basic Binning Example

Suppose a numerical feature **X** has values ranging from **15 to 425**. We create five bins:

Bin	Range
1	15 – 34
2	35 – 117
3	118 – 279
4	280 – 392

5	393 – 425

All values of X that fall within the same range are treated identically by the model.

---

4. One-Hot Encoding of Bins

Each bin is represented as a **one-hot vector** in the feature vector:

Bin	Feature Vector
1	[1.0, 0.0, 0.0, 0.0, 0.0]
2	[0.0, 1.0, 0.0, 0.0, 0.0]
3	[0.0, 0.0, 1.0, 0.0, 0.0]
4	[0.0, 0.0, 0.0, 1.0, 0.0]
5	[0.0, 0.0, 0.0, 0.0, 1.0]

Although X is a single column in the dataset, **binning expands it into multiple features**, each with its own learned weight.

---

5. Conceptual Diagram: Raw vs Binned Feature

```
Raw feature X:
15  22  29  36  78  145  300  410

Bins:
|----Bin1----|----Bin2----|----Bin3----|----Bin4----|--Bin5--|
15          34           117          279          392       425

Model sees:
Bin1 Bin2 Bin3 Bin4 Bin5
 1     0     0     0     0   ← X = 22
 0     1     0     0     0   ← X = 78
 0     0     1     0     0   ← X = 145
```

---

6. When Binning Is Better Than Scaling

Binning is a good alternative to scaling or clipping when:

- The linear relationship between feature and label is weak
- The feature exhibits **distinct clusters**
- The model should treat values within a range similarly

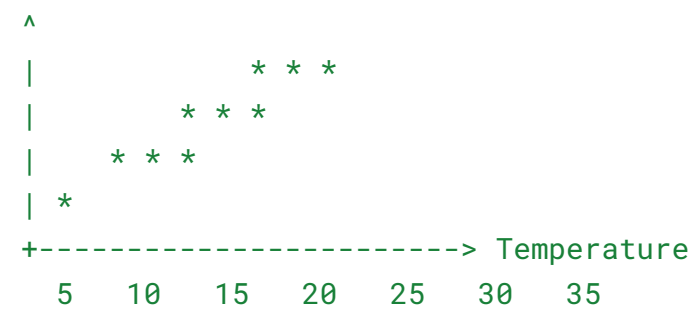
Although binning may seem counterintuitive (e.g., treating 37 and 115 identically), it often produces **more meaningful representations** when data is clumpy rather than linear.

Case Study: Temperature vs Number of Shoppers

7. Observed Pattern

Consider a dataset predicting the **number of shoppers** based on **outside temperature**.

Scatter Plot (Conceptual)



The data naturally clusters rather than forming a straight line.

---

8. Why Raw Temperature Is Suboptimal

If temperature is treated as a single numeric feature:

- A linear model assigns one weight
- A temperature of 35 has five times the influence of 7
- This assumption does not match observed behavior

---

9. Binning the Temperature Feature

Based on clustering, we define three bins:

Bin	Temperature Range
1	4 – 11
2	12 – 26
3	27 – 36

Each bin receives a separate learned weight.

Diagram

```
Temperature:
|--Cold--|------Comfortable------|--Hot--|
 4      11      12             26   27   36
```

This representation aligns better with human behavior and improves model performance.

---

10. Choosing the Number of Bins

Too many bins is usually a mistake:

- Each bin requires enough examples to learn from
- Excessive bins increase feature dimensionality
- Sparse bins lead to unreliable weights

Rule of thumb: **fewer, well-populated bins are better than many sparse bins**

Quantile Bucketing

---

11. Problem with Equal-Width Bins

Equal-width bins divide the feature range evenly, not the data density.

Example: Car Prices

```
Price Range ($) | # Cars
0 – 10k         | ████████████████
10 – 20k        | ████████
20 – 30k        | ████
30 – 40k        | ██
40 – 60k        | ▉
```

High-price bins have too few examples for effective learning.

---

12. Quantile Bucketing Solution

Quantile bucketing chooses bin boundaries so that **each bin contains approximately the same number of examples**.

Diagram Comparison

Equal-width bins

```
|----|----|----|----|----|----|
Lots        Fewer        Very few
```

Quantile bins

```
|--|--|----|------|-----------|
Same number of examples per bin
```

As a result:

- Some bins span narrow ranges
- Others span wide ranges
- Each bin is equally learnable

---

13. Benefits of Quantile Bucketing

- Ensures sufficient data per bin
- Reduces the influence of outliers

- Improves training stability
- Often superior for skewed distributions

---

14. When Not to Bin

Do **not** bin when:

- The feature-label relationship is mostly random
- No stable or interpretable grouping exists
- Binning introduces artificial patterns

In such cases, leaving the feature unbinned or excluding it may be preferable.

---

15. Summary of Binning Concepts

Concept	Key Idea
Binning	Converts numeric ranges into categories
One-hot encoding	Represents bins as separate features
Use cases	Nonlinear, clustered data
Too many bins	Causes sparsity and overfitting
Quantile bucketing	Equalizes data per bin
Trade-off	Precision vs learnability

---

16. Key Takeaways

- Binning allows models to learn **range-based patterns**
- It is especially effective when linear assumptions fail
- Quantile bucketing ensures statistical robustness
- Binning decisions must be driven by **data visualization and domain knowledge**

Binning is not a simplification—it is a **strategic representation choice** that often enables models to learn patterns that raw numerical values obscure.

Lecture: Scrubbing and Enhancing Numerical Data in Machine Learning

---

Part I: Scrubbing Numerical Data

1. Why Data Scrubbing Matters

In real-world machine learning, raw data is rarely clean. Much like apples harvested from an orchard, datasets contain a mixture of high-quality examples and unusable or misleading ones. Before a dataset is ready for training, significant effort must be spent **removing bad examples and repairing salvageable ones**.

Even a small number of bad data points can degrade model performance, bias predictions, or prevent convergence altogether.

---

2. Common Sources of Unreliable Data

Numerical datasets often contain unreliable examples due to the following issues:

Problem Category	Example
Omitted values	A census record missing a person's age
Duplicate examples	The same server logs uploaded twice
Out-of-range values	An extra digit accidentally typed
Bad labels	A maple tree mislabeled as an oak

---

3. Detectable Problems via Automation

Several data issues can and should be detected programmatically:

- **Missing (omitted) values**
- **Duplicate examples**
- **Out-of-range feature values**

For example:

- A dataset containing repeated rows should be deduplicated.
- If a temperature feature is defined to be between 10°C and 30°C, values outside this range should be flagged.

Automated checks form the first line of defense in data quality control.

---

4. Label Consistency and Human Bias

When labels are generated by multiple human annotators, inconsistencies often arise.

Recommended practice:

- Perform **statistical checks** to verify that different raters produce comparable label distributions.
- Identify raters who may be systematically harsher, more lenient, or using different criteria.

---

5. Fixing Bad Data

Once problematic examples are detected, they are typically handled in one of two ways:

1. **Removal**
   Used when data is clearly incorrect or irreparable.
2. **Imputation**
   Used when values are missing but can be reasonably inferred (e.g., mean or median substitution).

The choice depends on data volume, importance of the feature, and downstream model sensitivity.

Part II: Qualities of Good Numerical Features

---

6. Clear and Meaningful Feature Names

Features should be **immediately understandable** to any human on the project.

Poor practice:

```
house_age: 851472000
```

Good practice:

```
house_age_years: 27
```

Although models do not care about naming, humans do—and unclear features increase the risk of errors and misinterpretation.

---

7. Checked and Validated Before Training

Outliers are often caused by data collection errors rather than genuine phenomena.

Example:

```
user_age_in_years: 224   (invalid)
user_age_in_years: 24    (valid)
```

Data should always be validated against known constraints before training.

---

8. Sensible Feature Design and Avoiding “Magic Values”

A **magic value** is a special numeric value used to represent missing or undefined data in an otherwise continuous feature.

Poor practice:

```
watch_time_in_seconds: -1
```

This forces the model to interpret a nonsensical numeric meaning (e.g., negative time).

---

9. Recommended Approach for Missing Continuous Values

Use a **separate indicator feature**:

```
watch_time_in_seconds: 4.82
is_watch_time_defined: True

watch_time_in_seconds: 0
is_watch_time_defined: False
```

This approach:

- Preserves numerical meaning
- Explicitly signals missingness
- Improves model interpretability

---

10. Handling Missing Values in Discrete Numerical Features

For discrete numerical features with a finite set of values, represent missing data as an additional category.

Example:

```
{
  0: electronics,
  1: books,
  2: clothing,
  3: missing_category
}
```

The model learns a separate weight for missing values, which is often desirable.

Part III: Polynomial Transforms

---

11. Motivation for Polynomial Transforms

Some relationships in data are inherently **nonlinear**, even when using numerical features.

Consider two classes of data points that cannot be separated by a straight line but can be separated by a curve.

---

12. Limitation of Linear Models

A linear regression model with one feature follows the form:

y=wx+by = wx + by=wx+b

Such a model cannot separate data that follows a curved boundary.

---

13. Introducing Polynomial Transforms

A **polynomial transform** creates a new synthetic feature by raising an existing feature to a power.

For example:

x2x^2x2

This new feature is treated like any other input feature.

---

14. Linear Models with Nonlinear Power

By adding a polynomial feature, the model becomes:

y=w1x+w2x2+by = w_1x + w_2x^2 + by=w1x+w2x2+b

Despite the nonlinearity in the feature space, the model remains **linear in its parameters** and can still be trained using gradient descent.

This allows the model to learn curved decision boundaries without changing the training algorithm.

---

15. Choosing Polynomial Degrees

Often, domain knowledge guides the choice of exponent:

- Physics: squared terms (e.g., gravity, energy)
- Signal propagation: inverse-square laws
- Economics and growth models: cubic or higher-order terms

Unnecessary high-degree polynomials should be avoided, as they increase overfitting risk.

---

16. Polynomial Transforms and Normalization

Polynomial transforms often change the scale of a feature dramatically.

Best practice:

- Experiment with **normalizing transformed features**
- Normalization after transformation frequently improves model stability and performance

---

17. Relation to Feature Crosses

Polynomial transforms are related to **feature crosses**, which combine multiple features rather than raising a single feature to a power.

Both techniques expand the feature space to help models capture complex relationships.

Key Takeaways

- Scrubbing is essential to protect models from bad data
- Clear naming and validation prevent silent failures
- Magic values should be avoided in continuous features
- Polynomial transforms allow linear models to learn nonlinear patterns
- Feature engineering decisions must be driven by data quality, domain knowledge, and empirical testing