Lecture: Datasets – Data Characteristics

1. Introduction

In machine learning, **data is the foundation** of model performance. Even the most advanced algorithms will fail if trained on insufficient, noisy, or unreliable data. This lecture examines the key characteristics of datasets that directly affect model quality and reliability.

---

2. What Is a Dataset?

A **dataset** is a collection of examples used to train, evaluate, or test a machine learning model.

- Each **row** represents an example
- Each **column** represents a feature or a label

Common Dataset Formats

- Tables (CSV files, spreadsheets, database tables)
- Log files
- Protocol buffers
- Outputs from other machine learning systems

Regardless of format, a model can only learn from the data provided.

---

3. Types of Data in Datasets

Modern datasets often contain multiple data types, each requiring specialized preprocessing.

Common Data Types

- **Numerical data** (e.g., age, price, temperature)

- **Categorical data** (e.g., color, country, user type)
- **Text data** (words, sentences, documents)
- **Multimedia data** (images, audio, video)
- **Outputs of other ML systems**
- **Embedding vectors** (dense numerical representations)

Each data type must be handled appropriately to ensure effective learning.

---

4. Quantity of Data

Rule of Thumb

A model should train on **at least 10–100 times more examples than trainable parameters**. In practice, successful models often use far more data.

Key Observations

- Large datasets with **few features** often outperform small datasets with many features
- Simple models trained on massive datasets can be highly effective
- The required dataset size varies widely by problem:
    - Some tasks need only dozens of examples
    - Others require millions or even trillions

Transfer Learning

Good performance can sometimes be achieved with small datasets by adapting a model pretrained on large, similar datasets.

---

5. Data Quality and Reliability

Practical Definition of Quality

A dataset is **high quality** if it helps the model achieve its intended goal. A low-quality dataset inhibits learning.

Reliability

Reliable data is data you can **trust**. Models trained on reliable datasets are more likely to generalize well.

---

6. Common Sources of Unreliable Data

When assessing data reliability, consider the following questions:

- How frequent are **label errors**?
- How noisy are the feature values?
- Is the dataset properly filtered for the task?

Common Problems

- **Omitted values** (missing feature data)
- **Duplicate examples**
- **Incorrect feature values**
- **Incorrect labels**
- **Corrupted segments of data** (for example, during system outages)

Automation, such as schema validation and unit tests, is recommended to detect unreliable data.

---

7. Handling Outliers

Large or diverse datasets almost always contain outliers.

- Outliers are not necessarily errors
- They may represent rare but valid cases

Deciding how to handle outliers is a critical modeling decision and depends on the problem domain.

---

8. Complete vs. Incomplete Examples

Complete Examples

A **complete example** contains values for all features.

Incomplete Examples

An **incomplete example** has one or more missing feature values, which can degrade model performance if not addressed.

Do not train models directly on incomplete examples.

---

9. Strategies for Handling Missing Data

There are two primary strategies:

1. Deleting Incomplete Examples

Delete examples that contain missing values when:

- The dataset has enough complete examples
- The missing data is random and infrequent

Alternatively, consider removing a feature entirely if it is often missing and adds little value.

---

2. Imputing Missing Values

Imputation replaces missing values with reasonable estimates.

Common imputation methods:

- Mean imputation
- Median imputation

- Model-based imputation

Imputation should be **careful and informed**, not random.

---

10. Signaling Imputed Values to the Model

Imputed values are usually less reliable than real measurements.

Best practice:

- Add an **indicator feature** (Boolean column) that specifies whether a value was imputed

Example:

- `temperature`
- `temperature_is_imputed`

This allows the model to learn to trust real values more than imputed ones.

---

11. Imputation and Feature Scaling

When numerical features are normalized using **Z-scores**:

- The mean becomes 0
- Mean-imputed values therefore often equal 0

This makes mean imputation especially convenient in standardized datasets.

---

12. Choosing Between Deletion and Imputation

There is no universal rule.

If unsure:

1. Build one dataset with deleted incomplete examples
2. Build another dataset with imputed values
3. Train models on both
4. Compare performance

This empirical approach often yields the best decision.

---

13. Example: Imputation Exercise

Given a dataset sorted by time:

Timestamp	Temperature
June 8, 2023 09:00	12
June 8, 2023 10:00	18
June 8, 2023 11:00	missing
June 8, 2023 12:00	24
June 8, 2023 13:00	38

A reasonable imputed value is **23**, as it is consistent with neighboring values.

After imputation, the dataset should be **randomized** before training to avoid order-related bias.

---

14. Summary

Key takeaways from this lecture:

- Model performance depends heavily on dataset quality and quantity

- Datasets can contain multiple data types requiring different preprocessing
- Large, reliable datasets often outperform small, complex ones
- Incomplete examples must be handled through deletion or imputation
- Good imputation improves models; poor imputation harms them
- Always signal imputed values to the model when possible

---

15. Closing Remarks

High-quality data is not accidental—it is engineered. Careful attention to data characteristics is one of the most impactful investments you can make in any machine learning project.

Lecture: Datasets – Labels and Class Imbalance

1. Introduction

In supervised machine learning, **labels** define what the model is trying to predict. The quality, reliability, and distribution of labels strongly influence model performance. In this lecture, we examine:

- Direct vs. proxy labels
- Human-generated versus automated labels
- Class-balanced vs. class-imbalanced datasets
- Practical techniques for training on imbalanced data

---

2. What Is a Label?

A **label** is the target value a model is trained to predict.

- In classification, labels represent categories (for example, spam vs. not spam)

- In regression, labels represent continuous values (for example, house price)

Like features, labels must be represented as **floating-point values**, since machine learning models operate on numerical computations.

---

3. Direct Labels vs. Proxy Labels

Direct Labels

A **direct label** is exactly the value the model is trying to predict.

Example:

- Column: `bicycle_owner`
- Model goal: predict whether a person owns a bicycle

Direct labels are ideal and should be used whenever available.

Proxy Labels

A **proxy label** is an imperfect substitute for a direct label. It is correlated with the desired prediction but not identical.

Example:

- Column: `recently_bought_a_bicycle`
- Goal: predict bicycle ownership

Proxy labels are a **compromise**, but they are often necessary when direct labels are unavailable.

Choosing Between Direct and Proxy Labels

- Prefer direct labels whenever possible
- Use proxy labels only when necessary

- The usefulness of a proxy label depends on how strongly it correlates with the true prediction

Models trained on proxy labels are only as good as the relationship between the proxy and the true outcome.

---

4. Example: Evaluating a Proxy Label

Business goal:
Mail bicycle helmet coupons to bicycle owners.

Available label:
`recently_bought_a_bicycle`

This is a **good proxy label**, because most people who recently bought a bicycle now own one. However, it is not perfect—bicycles can be purchased as gifts or resold.

This example highlights the trade-off inherent in proxy labels.

---

5. Labels That Cannot Be Directly Represented

Sometimes a direct label exists conceptually but cannot be easily represented numerically.

Example:

- Subjective judgments (quality, intent, relevance)
- Complex human interpretations

In such cases, proxy labels are often the only practical solution.

---

6. Human-Generated Labels

What Are Human-Generated Labels?

Human-generated data is labeled by people rather than software.

Examples:

- Humans classifying images
- Linguists annotating text sentiment
- Meteorologists labeling cloud types

Advantages of Human-Generated Labels

- Humans can perform complex reasoning tasks
- Useful for nuanced or subjective labels
- Forces clear and consistent labeling criteria

Disadvantages of Human-Generated Labels

- Expensive and time-consuming
- Subject to human error and inconsistency
- Often requires multiple raters per example

Practical Considerations

When using human raters, consider:

- Required expertise or language skills
- Number of labeled examples needed
- Time constraints and budget

Always validate human-generated labels by reviewing a subset yourself and measuring agreement across raters.

---

7. Automated Labels

Automated labels are generated by software or other ML systems.

Examples:

- System logs
- Sensor readings
- Predictions from an existing model

Models can train on a mixture of automated and human-generated labels. However, maintaining multiple labeling pipelines can increase complexity.

---

8. Introduction to Class Imbalance

Many real-world classification problems suffer from **class imbalance**.

Definitions

- **Class-balanced dataset:** roughly equal numbers of each class
- **Class-imbalanced dataset:** one class appears far more frequently

In an imbalanced dataset:

- The **majority class** is the more common label
- The **minority class** is the rarer label

---

9. Real-World Examples of Class Imbalance

- Credit card fraud detection (fraud < 0.1%)
- Medical diagnosis of rare diseases (< 0.01%)
- Defect detection in manufacturing

Class imbalance is the norm, not the exception.

---

10. Why Class Imbalance Is Difficult

Training relies on batches containing enough examples of **both** classes.

In severely imbalanced datasets:

- Many batches contain no minority examples
- The model fails to learn what the minority class looks like
- Accuracy becomes a misleading metric

A model that always predicts the majority class may appear “accurate” but be useless.

---

11. Two Goals of Training

A well-trained model must learn:

1. The relationship between features and labels
2. The true distribution of classes

Standard training mixes these two goals, which causes problems in imbalanced datasets.

---

12. Technique: Downsampling and Upweighting

A two-step technique separates these goals:

1. **Downsample the majority class**
2. **Upweight the majority class during training**

This approach is counterintuitive but effective.

---

13. Step 1: Downsampling the Majority Class

Downsampling means removing many majority-class examples during training to create a more balanced dataset.

Example:

- Original dataset: 99% majority, 1% minority
- Downsample majority by factor of 25
- New training set: 80% majority, 20% minority

This ensures batches contain enough minority examples to learn effectively.

---

14. Step 2: Upweighting the Majority Class

Downsampling creates an artificial class distribution, introducing **prediction bias**.

To correct this:

- Multiply the loss of majority-class errors by the same factor used in downsampling

Example:

- Downsampling factor: 25
- Majority-class loss is multiplied by 25

This restores the model's understanding of the true class distribution.

---

15. Choosing Downsampling and Upweighting Factors

There is no universal rule.

- Treat downsampling and upweighting as **hyperparameters**
- Experiment with different values
- Evaluate using appropriate metrics (precision, recall, AUC)

---

16. Benefits of This Approach

Downsampling and upweighting provide:

- Better understanding of minority class patterns
- Preservation of real-world class probabilities
- Faster training convergence
- Improved overall model quality

---

17. Key Metrics Reminder

For class-imbalanced datasets:

- Accuracy is often misleading
- Prefer precision, recall, F1 score, and ROC-AUC

---

18. Key Terms

- Label
- Direct label
- Proxy label
- Majority class
- Minority class
- Class-balanced dataset
- Class-imbalanced dataset
- Downsampling
- Upweighting
- Prediction bias

- Batch
- Hyperparameter

---

19. Summary

In this lecture, we learned that:

- Labels define the learning objective
- Direct labels are ideal; proxy labels are often necessary
- Human-generated labels are powerful but costly
- Most real-world datasets are class-imbalanced
- Downsampling and upweighting enable effective training on imbalanced data

---

20. Closing Remarks

High-quality labels and thoughtful handling of class imbalance are critical to building reliable machine learning systems. Careful label design and training strategies often matter more than model complexity.

Lecture: Datasets – Dividing the Original Dataset

1. Introduction

In software engineering, testing is essential to ensure correctness. Machine learning is no different. A model that performs well only on the data it was trained on provides little real-world value. Therefore, **properly dividing datasets** is critical to evaluating whether a model truly generalizes.

This lecture explains:

- Why datasets must be split

- The roles of training, validation, and test sets
- Common pitfalls when evaluating models
- Best practices for dataset splitting

---

2. Why We Split Datasets

A machine learning model should be evaluated on **different examples** than those it was trained on.

Testing on the training data:

- Overestimates performance
- Encourages memorization rather than learning
- Provides weak evidence of real-world effectiveness

To obtain unbiased performance estimates, we split the original dataset into separate subsets.

---

3. The Naïve Two-Way Split

A common first idea is to split the dataset into:

- **Training set** (~80%) – used to train the model
- **Test set** (~20%) – used to evaluate performance

While this approach is better than no split at all, it has a serious flaw.

---

4. The Problem with Reusing the Test Set

Suppose you:

1. Train a model on the training set

2. Evaluate on the test set
3. Adjust hyperparameters or features based on test results
4. Repeat this process many times

What Goes Wrong?

Over time, the model indirectly **fits the test set**, even though it is not trained on it directly. This is analogous to “teaching to the test.”

As a result:

- Test performance becomes overly optimistic
- Real-world performance may degrade

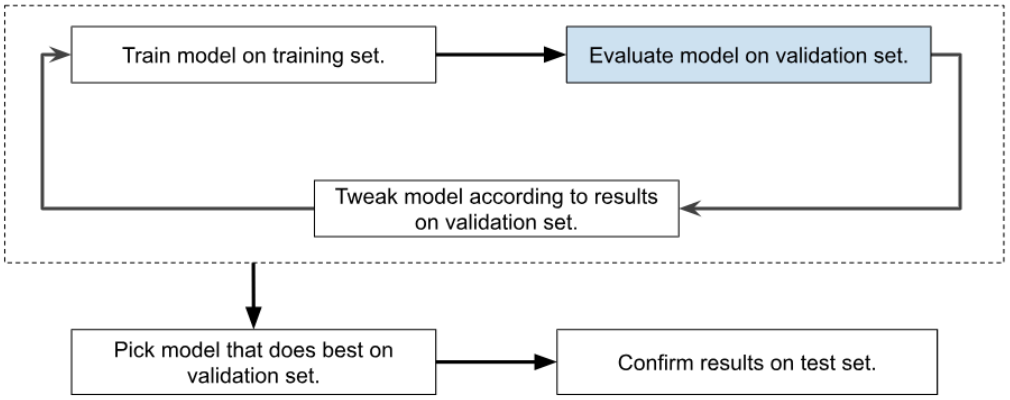
---

5. The Three-Way Split: Best Practice

To avoid this problem, divide the dataset into **three subsets**:

1. **Training set** (~70%)
   Used to train the model.
2. **Validation set** (~15%)
   Used during development to tune hyperparameters and features.
3. **Test set** (~15%)
   Used once at the very end to confirm final model performance.

This separation preserves the integrity of the test set.



---

6. Roles of Each Dataset

Training Set

- Fits model parameters
- Used repeatedly during optimization

Validation Set

- Guides model improvements
- Used to compare different models
- May be reused, but cautiously

Test Set

- Final, unbiased evaluation
- Should not influence model design
- Used as sparingly as possible

---

7. Recommended Workflow

A robust workflow follows these steps:

1. Train the model on the training set
2. Evaluate on the validation set
3. Adjust model, features, or hyperparameters
4. Repeat steps 1–3 until satisfied
5. Evaluate once on the test set to confirm results

Any feature transformations applied to the training set **must also be applied identically** to the validation set, test set, and real-world data.

---

8. Dataset Wear-Out

Even validation and test sets can “wear out.”

- Repeated reuse causes gradual overfitting
- Confidence in evaluation metrics decreases

Best practice:
Periodically collect new data to refresh validation and test sets. Starting over with fresh data often yields more reliable results.

---

9. A Common Pitfall: Duplicate Examples

If your test loss is unexpectedly low, investigate the data.

A frequent cause:

- **Duplicate examples** appearing in both training and test sets

This can happen when:

- Logs are duplicated
- Records are redundantly stored
- Data is split before deduplication

---

10. Why Duplicates Are Dangerous

If test examples duplicate training examples:

- The model is effectively tested on data it has already seen
- Performance metrics become meaningless

Rule:
After splitting the dataset, remove any validation or test examples that duplicate training examples.

---

11. Characteristics of a Good Test Set

A reliable test set must be:

- Large enough to produce statistically significant results
- Representative of the overall dataset
- Representative of real-world data
- Free of duplicates from the training set

A test set is only useful if it reflects the data the model will encounter in production.

---

12. The Zero-Sum Trade-Off

With a fixed dataset size:

> Every example used for testing is one less example used for training.

This is a fundamental trade-off:

- Larger training sets improve learning
- Larger test sets improve evaluation reliability

There is no perfect split—only informed compromises.

---

13. When Test Performance Looks Good but Reality Fails

If:

- Test loss is low

- Real-world performance is poor

Then the most likely issue is **dataset mismatch**.

Your dataset may:

- Be outdated
- Fail to capture real-world variability
- Reflect conditions that no longer apply

Action:
Analyze how real-world data differs and retrain using a more representative dataset.

---

14. How Large Should the Test Set Be?

There is no fixed percentage rule.

The test set should contain:

- Enough examples to yield **statistically significant** results

What qualifies as “enough” depends on:

- Task complexity
- Variance in predictions
- Evaluation metric used

Experimentation and domain judgment are required.

---

15. Key Terms

- **Training set** – data used to fit the model
- **Validation set** – data used for tuning and model selection

- **Test set** – data used for final evaluation
- **Hyperparameter** – a configuration value not learned during training
- **Generalization** – performance on unseen data

---

16. Summary

In this lecture, we learned that:

- Models must be evaluated on data not used for training
- A three-way split (training, validation, test) is best practice
- Reusing the test set causes overfitting
- Duplicate examples invalidate evaluation
- Dataset splitting is a zero-sum trade-off
- Real-world failure often indicates dataset mismatch

---

17. Closing Remarks

Proper dataset division is one of the most important—and most commonly mishandled—steps in machine learning. Careful splitting, disciplined evaluation, and periodic data refreshes are essential for building models that succeed beyond the lab.

Lecture: Datasets – Transforming Data and Generalization

1. Introduction

Machine learning models do not operate on raw data. Instead, they rely on **carefully transformed representations** of that data. The way features are transformed directly affects how efficiently a model trains and how well it generalizes to new, unseen examples.

In this lecture, we discuss:

- Why data transformation is necessary
- Common types of data transformations
- Sampling strategies when datasets are too large
- Privacy-aware filtering
- The relationship between data transformation and generalization

---

2. Why Data Transformation Is Necessary

Machine learning models can only train on **floating-point numerical values**. However, real-world datasets often contain:

- Strings
- Booleans
- Dates and timestamps
- Raw numerical values with very different scales

Before training, these features must be transformed into numerical representations that models can process effectively.

---

3. Transforming Non-Numerical Features

Categorical and Text Features

Many features, such as street names, product categories, or countries, are represented as strings.

Example:

- Raw feature: `"Broadway"`
- Problem: Models cannot train on strings
- Solution: Encode the string into a floating-point vector

Common approaches include:

- One-hot encoding
- Vocabulary encoding
- Embeddings

These techniques are covered in detail in the Categorical Data module.

---

4. Transforming Numerical Features (Normalization)

Even when features are already numerical, transformation is often still required.

Why Normalize?

Raw numerical features may:

- Have very large or very small values
- Operate on different scales
- Slow down or destabilize training

Normalization

Normalization transforms numerical values into a constrained range that improves training stability and convergence.

Common normalization techniques include:

- Min-max scaling
- Z-score normalization

Normalization is covered in detail in the Numerical Data module.

---

5. Consistency Across Datasets

All transformations applied to the training set **must also be applied identically** to:

- Validation set
- Test set
- Real-world inference data

Failure to apply consistent transformations leads to incorrect predictions and poor model performance.

---

6. Sampling When You Have Too Much Data

Some organizations have extremely large datasets that exceed computational or time constraints.

Why Sample?

- Training on all available data may be infeasible
- Diminishing returns beyond a certain dataset size
- Faster experimentation and iteration

Sampling Strategy

When selecting a subset:

- Prefer examples most relevant to the prediction task
- Preserve the underlying data distribution
- Avoid introducing unintended bias

Sampling should be deliberate and principled, not random truncation without analysis.

---

7. Filtering Personally Identifiable Information (PII)

High-quality datasets **exclude PII** such as:

- Names

- Addresses
- Phone numbers
- Email addresses

Why Filter PII?

- Protects user privacy
- Reduces legal and ethical risk
- Prevents models from learning sensitive or inappropriate associations

Filtering PII can affect model behavior, so its impact should be evaluated carefully.

---

8. Introduction to Generalization

Generalization refers to a model's ability to make accurate predictions on new, unseen data.

A model that performs well only on training data but poorly in the real world has **failed to generalize**.

---

9. How Data Transformation Affects Generalization

Proper data transformation improves generalization by:

- Reducing sensitivity to irrelevant scale differences
- Removing noise and inconsistencies
- Helping the model learn meaningful patterns instead of memorizing values

Poor transformation can cause:

- Overfitting
- Training instability

- Poor real-world performance

---

10. Overfitting and Memorization

Without proper transformation:

- Models may memorize rare or extreme values
- Models may rely on irrelevant numerical magnitudes
- Performance appears good in training but fails in deployment

Transformations such as normalization and categorical encoding help the model focus on structure rather than raw representation.

---

11. Data Distribution and Generalization

For good generalization:

- Transformed training data must resemble transformed real-world data
- Sampling and filtering must preserve real-world distributions

A mismatch between training and deployment data distributions is a common cause of model failure.

---

12. Practical Best Practices

- Transform all features into floating-point representations
- Normalize most numerical features
- Encode categorical features appropriately
- Apply transformations consistently across all datasets
- Sample large datasets thoughtfully

- Remove or anonymize PII
- Continuously evaluate generalization performance

---

13. Summary

In this lecture, we covered:

- Why data transformation is required in machine learning
- Encoding non-numerical features into floating-point values
- Normalizing numerical features for better training
- Sampling strategies for very large datasets
- Filtering PII for privacy and safety
- The central role of data transformation in model generalization

---

14. Closing Remarks

Data transformation is not a cosmetic preprocessing step—it is a foundational component of successful machine learning systems. Thoughtful transformations enable efficient training, protect user privacy, and, most importantly, help models generalize to the real world.