جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Lecture: (2)

Data load and Splitting for ML Models

Subject: Machine Learning
Class: Third
Lecturer: Dr. Maytham N. Meqdad

Data load and Splitting for ML Models

Splitting facts for system mastering models is an crucial step within the version improvement process. It includes dividing the to be had dataset into separate subsets for education, validation, and trying out the version. Here are a few common processes for splitting data:

1. Train-Test Split: The dataset is divided right into a training set and a trying out set. The education set is used to educate the model, even as the checking out set is used to assess the model's overall performance. The regular cut up is 70-eighty% for training and 20-30% for checking out, but this may vary depending on the scale of the dataset and the precise use case.

2. Train-Validation-Test Split: The dataset is split into three subsets - a schooling set, a validation set, and a trying out set. The training set is used to train the version, the validation set is used to tune hyperparameters and validate the version's overall performance for the duration of training, and the testing set is used to evaluate the very last version's overall performance.

3. K-fold Cross Validation: The dataset is divided into ok equally sized folds, and the version is educated and evaluated okay instances. Each time, k-1 folds are used for training, and 1-fold is used for validation/testing. This allows in acquiring greater strong overall performance estimates and reduces the variance in version evaluation.

4. Stratified Sampling: This technique guarantees that the distribution of training or other essential features is preserved in the training and trying out units. This is in particular beneficial when coping with imbalanced datasets, wherein some classes may additionally have only a few sample.

5. Time-primarily based Split: When coping with time collection facts, consisting of stock costs or weather statistics, the dataset is regularly cut up into schooling and checking out sets based on a chronological order. This facilitates in comparing the model's performance on future unseen facts.

It's vital to carefully keep in mind the information splitting approach primarily based at the particular hassle, dataset size, and other elements to make certain that the version is skilled and evaluated effectively. Proper statistics splitting enables in assessing the model's overall performance correctly and facilitates save you overfitting or underfitting of the version.

Some tips to choose Train/Dev/Test sets

The size of the train, dev, and test sets remains one of the vital topics of discussion. Though for general Machine Learning problems a train/dev/test set ratio of 80/20/20 is acceptable, in today's world of Big Data, 20% amounts to a huge dataset. We can easily use this data for training and help our model learn better and diverse features. So, in case of large datasets (where we have millions of records), a train/dev/test split of 98/1/1 would suffice since even 1% is a huge amount of data.

Python Program: Data Loading and Splitting for ML Models

```
# Import required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Step 1: Load the dataset
# (Example using a CSV file)
data = pd.read_csv("data.csv")

# Step 2: Separate features (X) and target variable (y)
X = data.drop("target", axis=1)   # Replace "target" with your column name
y = data["target"]

# Step 3: Split data into Training (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.30,
    random_state=42,
    stratify=y  # Keeps class distribution balanced
)

# Step 4: Split Temp into Validation (15%) and Test (15%)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.50,
    random_state=42,
    stratify=y_temp
)

# Step 5: Feature Scaling (important for many ML models)
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

```
X_test = scaler.transform(X_test)

# Print dataset shapes
print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", X_test.shape)
```

Explanation of the Steps for Python Program

1 Importing Libraries

The program begins by importing essential libraries:

- **pandas** for data manipulation and loading datasets.
- **train_test_split** from scikit-learn for splitting the dataset.
- **StandardScaler** for feature normalization.

---

2 Loading the Dataset

The dataset is loaded from a CSV file using:

```
pd.read_csv("data.csv")
```

This step reads the data into a pandas DataFrame for further processing.

---

3 Separating Features and Target Variable

Machine learning models require:

- **X (Features):** Independent variables used for prediction.
- **y (Target):** Dependent variable to be predicted.

We remove the target column from the dataset to create X and assign it separately to y.

---

4 First Data Split (Training + Temporary Set)

The dataset is split into:

- 70% Training data
- 30% Temporary data

The `stratify=y` parameter ensures that class distribution remains balanced, which is especially important in classification tasks.

The `random_state=42` ensures reproducibility.

---

5 Second Data Split (Validation + Test Sets)

The temporary dataset (30%) is further divided into:

- 15% Validation set
- 15% Test set

The validation set is used for tuning model hyperparameters, while the test set is reserved for final model evaluation.

---

6 Feature Scaling

Standardization is applied using `StandardScaler`:

- The scaler is **fitted only on the training data**.
- The same transformation is applied to validation and test data.

This prevents **data leakage** and ensures fair evaluation.

---

7 Output

Finally, the shapes of the datasets are printed to confirm the correct split proportions.

Fundamentals of Data Loading in Machine Learning

Data loading represents far more than simply reading files into memory—it encompasses a sophisticated orchestration of processes that directly influence training efficiency, model convergence, and ultimately, the quality of learned representations. Understanding these fundamentals is essential for any practitioner seeking to build production-grade machine learning systems.

The modern data loading pipeline must handle diverse challenges: efficiently managing datasets that exceed available memory, preprocessing raw data into model-ready formats, implementing intelligent batching strategies, and coordinating across distributed computing environments. Each decision in this pipeline creates ripple effects throughout the entire training process.

1. **Efficiency**

Optimised reading and preprocessing accelerate training cycles and reduce computational costs

2. **Scalability**

Distributed architectures enable training on datasets too large for single-machine processing

3. **Flexibility**

Modern loaders support dynamic data mixing and multi-source integration seamlessly

What is Data Loading?

Data loading constitutes the systematic process of efficiently reading, transforming, and preparing data for machine learning model training. This critical pipeline component encompasses multiple sophisticated operations: batching samples into appropriately sized groups for gradient computation, shuffling data to prevent order-dependent biases, and seamlessly handling datasets that originate from multiple heterogeneous sources.

The implementation of an efficient data loading strategy directly impacts two fundamental aspects of model development: training speed and convergence behavior. A well-designed loader ensures that the GPU or TPU never sits idle waiting for data, maintains consistent memory utilization, and presents samples in an order that facilitates effective learning. Poor loading strategies, conversely, can create bottlenecks that waste expensive computational resources and extend training times from hours to days.

1. Data Reading: - Efficiently access files from disk, cloud storage, or databases
2. Preprocessing;-Transform raw data into model-compatible formats with normalization and augmentation
3. Batching: - Group samples intelligently for optimal gradient computation
4. Delivery: - Stream prepared batches to training hardware without bottlenecks

Challenges in Data Loading at Scale

As machine learning models grow increasingly sophisticated and datasets expand exponentially, the challenges associated with data loading intensify dramatically. Modern foundation models like GPT-4 or Stable Diffusion require training on terabytes or petabytes of data—volumes that dwarf the capabilities of traditional single-machine loading approaches. These scale-related challenges manifest across multiple dimensions, each requiring thoughtful architectural solutions.

1. **Distributed Loading Complexity**

Large datasets demand parallel loading strategies across multiple workers, introducing synchronization challenges and potential race conditions. Coordinating dozens or hundreds of data loading processes requires sophisticated orchestration to maintain training efficiency.

2. **Workload Imbalance**

Non-uniform sample distributions cause some workers to finish early whilst others lag behind, creating idle time and resource waste. Dynamic load balancing becomes essential but introduces additional complexity.

3. **Memory Overhead**

Redundant file access states and duplicated metadata across workers consume precious memory resources. Multi-source datasets exacerbate this problem, potentially overwhelming available RAM and forcing expensive disk swapping.

Data Splitting – The Cornerstone of Reliable ML Evaluation

If data loading provides the infrastructure for efficient training, data splitting constitutes the methodological foundation for honest evaluation. The manner in which we partition our datasets into training, validation, and test sets fundamentally determines whether our models genuinely learn generalizable patterns or merely memories training examples. This seemingly simple act of division carries profound implications for model reliability and real-world performance.

The splitting process must navigate a delicate balance: providing sufficient training data for the model to learn rich representations whilst reserving adequate samples for unbiased evaluation. Poor splitting decisions create insidious problems that may only become apparent after expensive deployment, when models fail catastrophically on real-world data that differs subtly from their training distribution.

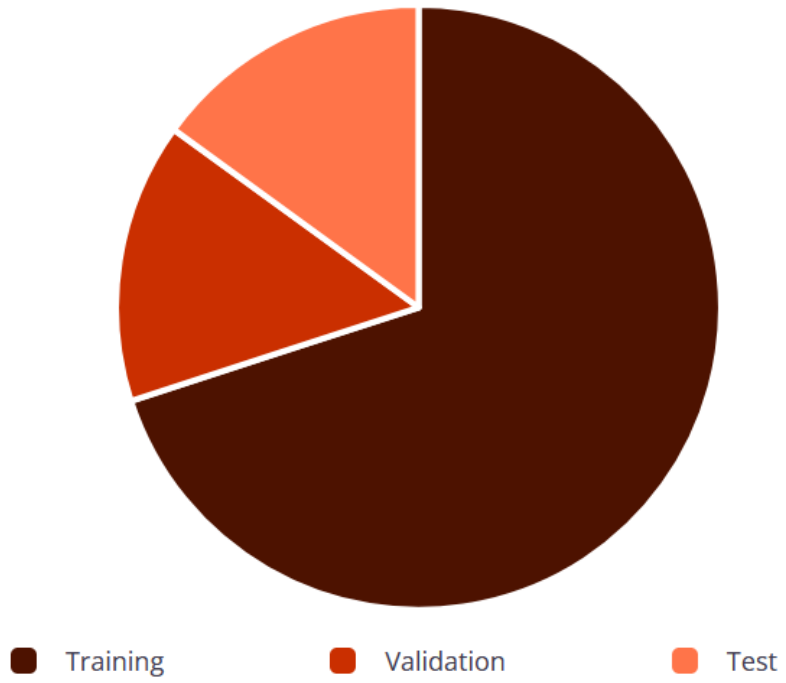
1. **Raw Dataset: -** Complete collection of labelled samples
2. **Strategic Split: -** Careful partitioning based on data characteristics
3. **Reliable Evaluation: -** Trustworthy performance metrics on held-out data

Purpose of Data Splitting

Data splitting serves as the primary mechanism for ensuring unbiased model evaluation and effective hyperparameter tuning. By partitioning available data into distinct subsets—typically training, validation, and test sets—we create the conditions necessary for honest assessment of generalization capabilities.

The training set provides examples from which the model learns patterns and relationships. The validation set enables iterative refinement, allowing us to adjust hyperparameters and architectural choices whilst monitoring performance on unseen data. Finally, the test set offers a definitive evaluation conducted only after all development decisions are finalized, providing our most trustworthy estimate of real-world performance.

Typical splitting ratios allocate approximately 70% of data for training, 15% for validation, and 15% for testing, though these proportions should adapt based on problem-specific considerations such as dataset size, complexity, and domain requirements.



This distribution represents a common baseline, but should be adjusted based on dataset size and problem requirements. Smaller datasets may require different ratios or cross-validation approaches.

Practical Tips for Effective Data Splitting and Loading

Whilst theoretical frameworks and cutting-edge research provide essential foundations, successful machine learning practice ultimately depends on implementing robust, practical workflows for data splitting and loading. The gap between knowing best practices and consistently applying them in production environments remains substantial. This chapter distils hard-won lessons from practitioners who have deployed ML systems at scale, transforming abstract principles into concrete, actionable guidelines.

The recommendations that follow represent battle-tested wisdom derived from countless projects across diverse domains. They address common pitfalls that ensnare even experienced practitioners: subtle forms of leakage that slip through validation, performance bottlenecks that emerge only at scale, and evaluation biases that mislead development efforts. By incorporating these practices into your standard workflow, you can avoid expensive mistakes and build models that perform reliably when stakes are highest.

01

Analyze Data Structure

Identify relationships, hierarchies, and potential leakage pathways before splitting

02

Select Appropriate Strategy

Choose splitting method based on data characteristics and deployment scenario

03

Implement Rigorous Validation

Verify split quality and independence through statistical tests

04

Document Decisions

Record splitting rationale and parameters for reproducibility

References

[1] Machine Learning Bookcamp, Alexey Grigorev.
[2] Python Data Science Handbook, Jake VanderPlas
[3] https://github.com/microsoft/Data-Science-For-Beginners
[4] geeks for geeks: https://www.geeksforgeeks.org/