Data Sampling and Model Selection

1. Sampling

Sampling determines how data points are selected and allocated for model training, validation, and testing. Proper sampling ensures that the model generalises well and that performance metrics are reliable. The choice of sampling technique depends on the target feature characteristics, dataset size, distribution, and the overall machine learning objective.

1.1 Sampling Techniques

- **Random Sampling:** Random sampling assigns data points to the train, validation (dev), and test sets completely at random. This is the most commonly used approach and is suitable when the data is well-distributed and free from significant temporal or categorical imbalance.

- **Stratified Sampling:** Stratified sampling is used when the dataset is imbalanced or when the target variable contains multiple classes with uneven representation. It ensures that the proportion of each target class remains consistent across the train, dev, and test splits, thereby preventing bias and improving the reliability of performance evaluation.

  Stratification Target: The stratification target refers to the variable on which stratification is performed. Typically, this is the target feature whose category ratios need to be preserved across dataset splits.

- **Top Sampling:** Top sampling is generally used in time-series or sequential datasets. It involves training on earlier data points and evaluating on later data points to test how well the model generalises into the future. The order is strictly based on the order of rows in the dataset, not on any specific feature value. In other words, the split respects the dataset's existing sequence (usually chronological order) rather than sorting or grouping by a particular column. This method preserves temporal order and prevents data leakage, which is critical for forecasting tasks.

> **Note**
> Other specialized sampling techniques also exist in machine learning and statistics, such as **Latin Hypercube Sampling (LHS)** (used in simulation and experimental design for better parameter space coverage), as well as **undersampling** and **oversampling** (used primarily for handling class imbalance).

1.2 Recommended Data Split Ratios

The recommended train / dev / test split varies by dataset size, as outlined in the table below.

Dataset Size	Train	Dev	Test
> 1 Million	99.5%	0.4%	0.1%
100K – 1 Million	98%	1%	1%
10K – 100K	80–90%	5–10%	5–10%
1K – 10K	70–80%	10–15%	10–15%
< 1,000	70%	15%	15%

Note

If the dataset is extremely small, the dev set may be omitted and only a train–test split is used.

1.3 Training Batch Size Strategy

Batch size plays a key role in model optimisation, stability, and computational efficiency. The choice depends on dataset size and hardware constraints.

- **Full-Batch Gradient Descent:** Processes the entire training dataset in a single update step. Useful for small to medium datasets, offering highly stable updates but often slower due to large computation per step.
- **Mini-Batch Gradient Descent:** Divides the training data into smaller batches (commonly multiples of 32 — e.g. 32, 64, 128). This is the standard approach for large datasets because it balances computational efficiency and stability.
- **Stochastic Gradient Descent (SGD):** Updates model parameters using one data point at a time. Useful for small datasets and optimisation problems requiring exploration, as it helps escape local minima — but results in noisy updates.

⚠ Note — Batch Size Does Not Apply to Tree-Based Models

The batch size strategies above apply exclusively to gradient-descent-based models (e.g. linear/logistic regression and neural networks). They do **not** apply to **Decision Trees, Random Forests, or Gradient Boosting** models.

Tree-based algorithms learn by recursively partitioning the feature space using splitting criteria (e.g. Gini impurity, information gain, or gradient residuals). They process the full training dataset — or a bootstrapped subset — during each split evaluation, and the concept of iterative mini-batch parameter updates through gradient descent does not apply to their learning procedure. Consequently, there is no batch size hyperparameter to configure for these models.

2. Model Selection Guidelines

Selecting the right algorithm is one of the most consequential decisions in machine learning. The table below provides practical guidance on when to use each of the available model families, along with conditions where a different approach would be preferable.

2.1 Algorithm Quick-Reference

Algorithm	Task Type	Best When	Avoid When
Linear Regression	Regression	Target is continuous; relationship between features and target is approximately linear; interpretability is critical; baseline benchmarking	Strong non-linearity exists; complex feature interactions are present
Logistic Regression	Classification	Binary or multi-class output; need probability estimates; data is linearly separable; fast training and inference required	Highly non-linear boundaries; very large feature spaces with interactions
Decision Tree	Both	Need a fully interpretable, rule-based model; mixed data types (numerical & categorical); quick prototyping	Data is high-dimensional; generalisation is a priority (prone to overfitting without pruning)
Random Forest	Both	Medium-to-large datasets; high dimensionality; need robust generalisation with minimal tuning; feature importance ranking	Extremely large datasets where inference latency matters; full model transparency is required
Gradient Boosting	Both	Structured/tabular data; highest predictive accuracy is the goal; handles missing values and mixed types well	Very small datasets (risk of overfitting); real-time low-latency applications without optimisation
Neural Network (MLP)	Both	Large datasets; complex non-linear patterns; high-dimensional inputs (images, text embeddings, sensor data)	Small datasets; high interpretability required; limited compute resources; tabular data where tree models may suffice

2.2 Detailed Overview

Linear & Logistic Regression

Linear and logistic regression are the foundational starting point for regression and classification tasks respectively. They are highly interpretable, fast to train, and work well when the relationship between features and the target is approximately linear. Logistic regression outputs calibrated probabilities, making it suitable for risk-scoring applications. Both models serve as strong baselines before introducing more complex alternatives.

- Use as a baseline before testing non-linear models.
- Regularisation (L1 / L2) should be applied when features number in the hundreds or thousands.

Decision Trees

Decision trees produce explicit, human-readable rule sets that are straightforward to explain to non-technical stakeholders. They handle both numerical and categorical features natively and require minimal preprocessing. However, unpruned trees tend to overfit aggressively, and their instability (high variance) makes them less reliable than ensemble counterparts (such as Random Forest or Gradient Boosting) on most real-world datasets.

- Best suited to scenarios where rule transparency is non-negotiable.
- Apply max-depth or min-samples-leaf constraints to control overfitting.
- Consider Random Forest or Gradient Boosting when predictive performance matters more than interpretability.

Random Forest

Random Forest is an ensemble of decision trees trained on bootstrapped samples with random feature subsets at each split. This combination of bagging and feature randomisation substantially reduces variance without significantly increasing bias. Random Forests are robust to outliers, handle high-dimensional data well, and provide reliable feature importance scores making them one of the most versatile algorithms for structured/tabular data.

- Strong default choice for medium-to-large structured datasets.
- Parallelises well across CPU cores; less memory-efficient than single trees.
- Feature importances from Random Forest are a useful starting point for feature selection.

Gradient Boosting

Gradient Boosting builds an ensemble sequentially, where each new tree corrects the residual errors of its predecessors. This sequential correction mechanism typically yields the highest predictive accuracy among tree-based methods on structured data.

- Preferred when maximising accuracy on structured/tabular data is the primary objective.
- Learning rate, tree depth, and number of estimators are the most impactful hyperparameters to tune.
- Early stopping on a validation set is strongly recommended to prevent overfitting.

Neural Networks (Multi-Layer Perceptron)

Neural networks are universal function approximators capable of learning highly complex, non-linear relationships from raw data. They excel when datasets are large, features are high-dimensional or unstructured (e.g. embeddings, sensor streams), and sufficient compute is available. Their performance scales favourably with data volume, but they require careful regularisation, longer training times, and are significantly harder to interpret than tree-based models.

- Preferred for large datasets (typically > 100K samples) with complex feature interactions.
- Require feature scaling (standardisation or normalisation) as a preprocessing step.
- For purely tabular/structured data, ensemble tree methods frequently match or exceed neural network performance with far less tuning effort.
- Dropout, batch normalisation, and learning-rate scheduling are recommended regularisation strategies.

For detailed technical specifications including model formulation, hyperparameters, optimization strategies, and convergence controls, please refer to the complete model documentation for the given model type.

2.3 Model Selection Decision Flow

Use the following decision flow as a starting heuristic before running systematic experiments:

1. **Start with Linear / Logistic Regression** as a baseline for any regression or classification task.
2. If baseline performance is insufficient and data is structured/tabular, **try Random Forest** for a robust, low-tuning alternative.
3. If maximum accuracy on tabular data is required, **escalate to Gradient Boosting** hyperparameter tuning.
4. If interpretability is the top priority and data volume is moderate, consider a **Decision Tree** with appropriate pruning.
5. **Reserve Neural Networks** for large datasets, unstructured inputs, or when ensemble methods have plateaued.

Best Practice

Always establish a simple baseline (linear/logistic regression) before investing in complex models. Complexity should be introduced incrementally, justified by measurable performance gains on the validation set.

Run experiments using consistent train / dev / test splits (see Section 1.2) to ensure fair comparison across model families.