**Course Title:** Environmental Data Analytics.
**Degree Program**: Masters in data science.
**Instructor:** Mohammad Mahdi Rajabi

UNIVERSITÉ DU LUXEMBOURG

# Automated Machine Learning

## What is AutoML?

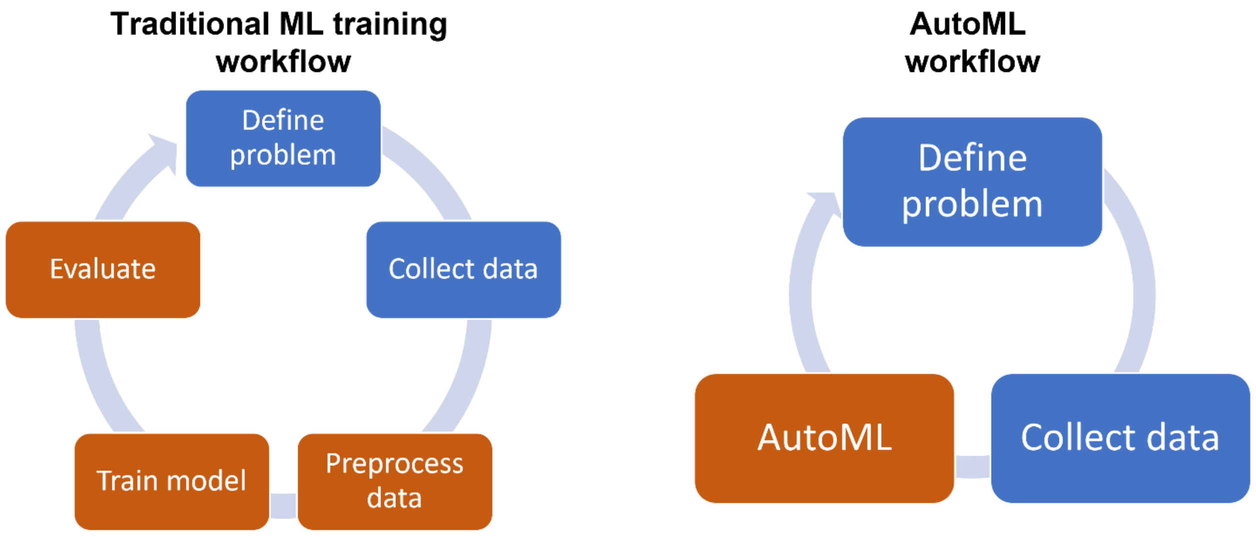
Machine learning workflows involve **trial-and-error at each step**, including data preprocessing, feature engineering, model selection, hyperparameter optimization, and model evaluation. Since these steps are interconnected, they collectively result in a significant amount of manual effort. This process often requires substantial domain knowledge to make informed decisions at each stage. Consequently, practitioners face a major challenge: either invest a lot of time and resources to thoroughly explore all possible options or rely on initial guesses, heuristics, or best judgments, potentially overlooking better-performing alternatives.

**Automated Machine Learning (AutoML)** refers to systems or tools designed to handle the entire machine learning pipeline automatically, from raw data input to producing a ready-to-deploy model. The goal is to **reduce the manual effort** and expertise required for trial-and-error processes in building machine learning models, making them more accessible to a broader range of users while also saving time for experienced practitioners.

AutoML is not just about automation; it also focuses on **optimizing resources** and minimizing the need for extensive trials. **AutoML systems avoid brute-force search** due to the sheer size and complexity of the search space involved in machine learning tasks. The combinations of algorithms, hyperparameters, preprocessing steps, and feature engineering pipelines grow exponentially with even modest choices. Evaluating each configuration involves training a model, which can be computationally expensive, especially with large datasets or complex models. Since not all configurations are equally promising—many are redundant or suboptimal—brute-force exploration is wasteful. Instead, AutoML employs intelligent optimization techniques to efficiently navigate the search space, delivering reliable insights without the impracticality of exhaustive exploration.

**AutoML systems are complex services** that are typically developed with significant effort, often by organizations or researchers, and made available to others either as free, open-source tools or as paid services. These systems streamline the machine learning process, making advanced ML techniques accessible to a broader audience.

The challenge of creating an AutoML service stems from the complexity of automating a process that must work across diverse data types and machine learning tasks, handle numerous decisions throughout the workflow automatically, and remain computationally efficient and manageable.



*AutoML vs Traditional Machine Learning.*

## Why is AutoML important for environmental data analytics?

AutoML is important in every application of machine learning, but it holds particular significance in the field of environmental data analytics. This is due to two key reasons:

1. **Challenges in Environmental Data and ML Tasks**: Environmental data is often highly complex, requiring advanced feature engineering and sophisticated model architectures. This complexity makes the manual trial-and-error process of building models more time-consuming and computationally demanding.
2. **Bridging the Expertise Gap**: There is limited overlap between environmental experts and machine learning specialists. AutoML addresses this gap by enabling less experienced environmental professionals to carry out advanced machine learning tasks, reducing reliance on extensive technical expertise and allowing for broader application of ML in environmental research.

AutoML is closely related to several key concepts, though not all AutoML systems include all of them. These concepts represent stages or capabilities that can be part of an automated machine learning workflow, depending on the system's focus and design.

## Automate Feature Engineering

**Automated Feature Engineering** is the process of using algorithms to **automatically** create, transform, and select features from raw data for machine learning models.

Automated Feature Engineering is built upon **three pillars**:

1. **Algorithmically synthesizing new features** to augment the raw set of features. This involves two main approaches:
   a) **Abstraction**: Transforming individual features to extract new insights or improve usability. Examples includes Scaling (Normalizing values to a common range, e.g., 0–1); Log Transformations (Reducing skewed data by applying logarithms); and Categorical Encoding (Converting categories into numerical formats, e.g., one-hot encoding).
   b) **Combination**: Generating new features by combining existing ones to reveal relationships or interactions. Examples include Feature Interactions (Creating products or ratios, e.g., "Price × Quantity" = "Revenue"); Aggregations (Summarizing data, e.g., calculating averages per group); Polynomial Features (Capturing non-linear relationships by raising features to powers); and Distance Measures (Computing distances between points in multidimensional or geospatial data).

   The process of automating feature creation typically involves algorithms going through a **predefined list of transformations and combinations** to generate new features, as this is perhaps the simplest approach. However, this method can be wasteful in terms of resources. To address this inefficiency, **dynamic methods** are often considered more effective. Dynamic methods can be categorized into two groups:
   a) **Generate a customized list of transformations and combinations** based on the specific dataset (e.g., data types, statistical properties, or domain-specific rules) and then systematically apply them.
   b) **Create new features sequentially**, where the results from analyzing one feature guide the choice of the next feature to generate. This includes the use of methods such as Reinforcement Learning (RL), Greedy Forward Selection, Iterative Feature Refinement (IFR) and Genetic algorithm.
2. **Metrics to evaluate and compare feature sets**, to assess the quality, relevance, and importance of features or feature sets with respect to the target variable. They help identify the most meaningful features and discard redundant or irrelevant ones. These metrics can be categorized into three groups:
   a) **Statistical Metrics**: Assess the direct relationship between features and the target variable in a supervised setting using statistical measures. Examples include the **Correlation Coefficient** (linear relationships) and **Mutual Information** (non-linear dependencies).
   b) **Model-Based Metrics**: Use auxiliary, often classical machine learning models, to assess feature importance. For example, tree-based models like **Random Forest** or **Gradient Boosted Trees** assign importance

scores to features based on their contribution to reducing errors during training.

c) **Predictive Metrics**: Evaluate feature sets by directly measuring their effect on the performance of the main model. These methods are effective but computationally expensive, as they require training the model. Examples of evaluation criteria include **Cross-Validation Accuracy/Score**, **Adjusted R²** (for regression), and **AUC-ROC** (for classification).

3. **Algorithms to systematically search the feature space**, optimizing the selection of features or feature sets for the given task.

   The feature space for this search includes all potential features that could be incorporated into the model. This consists of **Original features** derived directly from the raw dataset, and **Engineered features**, which are generated through transformations or combinations of the original features. Feature space search methods include:

   a) **Exhaustive Search**: Evaluates all possible feature combinations to find the best subset; accurate but computationally expensive.

   b) **Randomized Search**: Randomly samples feature subsets for evaluation; faster but may miss optimal sets.

   c) **Heuristic-Based Search**: Guides the search with rules to reduce computational cost. Examples include **Greedy Forward Selection**, which adds features incrementally based on performance, and **Greedy Backward Elimination**, which removes features iteratively starting from the full set.

   d) **Evolutionary Algorithms**: Iteratively evolve feature subsets inspired by natural selection. Examples include **Genetic Algorithms (GAs)** and **Particle Swarm Optimization (PSO)**.

   e) **Bayesian Optimization**: Builds probabilistic models to efficiently explore high-dimensional feature spaces.

## Model Selection

A critical and often challenging step in machine learning is selecting the right model type and architecture that best suits the specific task at hand. With numerous model types and architectural variations, finding the optimal configuration among countless possibilities requires careful evaluation. This process significantly impacts performance, as the chosen model must balance accuracy, efficiency, and task-specific constraints to achieve the best results.

In the context of AutoML, this challenge can be addressed either through **Automated Model Selection**, which applies to a range of model types, or, if the focus is specifically on neural networks, through **Neural Architecture Search (NAS)**. Both often need to be performed in conjunction with **Hyperparameter Optimization (HPO)** to fine-tune the selected models or architectures for optimal performance.

## Automated Model Selection

Model selection is the process of choosing the best-performing model from a **set of predefined models** or configurations. It involves evaluating multiple existing models and comparing their performance on a given dataset. The candidate models are typically **predefined by the practitioner**, and the selection is carried out through systematic evaluation (often exhaustive or brute force evaluation), with early termination based on performance thresholds to save computational resources.

- **Early termination** in model selection refers to stopping the evaluation of a model or configuration before it completes its full process if it becomes evident that it is unlikely to outperform other candidates. This technique helps save computational resources by avoiding unnecessary evaluations of sub-optimal models. Early termination can be achieved through various methods, including:
    1. **Performance Thresholds**: Stop training if performance metrics (e.g., validation loss) fail to improve after a set number of iterations.
    2. **Budget Constraints**: Halt models exceeding allocated resources, such as time or computational cost, without showing promising results.
    3. **Confidence Intervals**: Use statistical methods to stop evaluations when the likelihood of outperforming the best model so far is low.

## Neural Architecture Search (NAS)

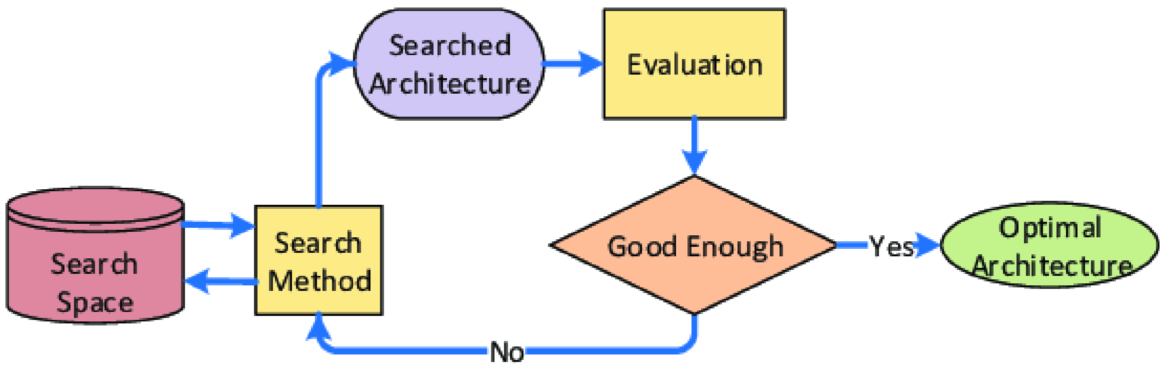
Neural Architecture Search (NAS) is a systematic method for creating and evaluating new neural network architectures to find the best one for a given problem. It often incorporates Hyperparameter Optimization (HPO) as part of the process to fine-tune the architectures for optimal performance. NAS is highly automated, often requiring minimal manual intervention beyond defining the search space and objective.

The NAS process involves the following key steps:

1. **Defining the Search Space**: The search space specifies the set of all possible neural network architectures that can be explored. This includes options like the number and type of layers (e.g., convolutional, recurrent), their connections (e.g., sequential, skip connections), activation functions, and more.

A common approach in NAS is **cell-based methods**, where architectures are constructed by assembling **small, modular components called cells**. A cell typically represents a specific layer or group of layers with defined operations (e.g., convolutions, pooling, skip connections). This approach makes the search space more manageable and scalable for complex architectures by focusing on all possible configurations of these cells, including the operations within each cell and the connections between them. Once the cells are defined, the NAS process centers on how to combine and stack these cells to form the overall network architecture.

2. **Search Strategy**: A search strategy is then used to navigate the defined space and identify promising architectures. Popular strategies include:
    a) **Reinforcement Learning (RL)**: Treats the search process as a decision-making problem where actions correspond to choosing architecture components.
    b) **Evolutionary Algorithms**: Simulates biological evolution by mutating and combining architectures to find better candidates.
    c) **Gradient-Based Methods**: Utilizes gradients to optimize the architecture directly, often leading to faster convergence.
3. **Evaluation Metric**: An evaluation metric, such as validation accuracy, loss, or computational efficiency, guides the search process by assessing the performance of candidate architectures. This feedback is used to refine the search strategy iteratively.



*Schematic of Neural Architecture Search (NAS).*

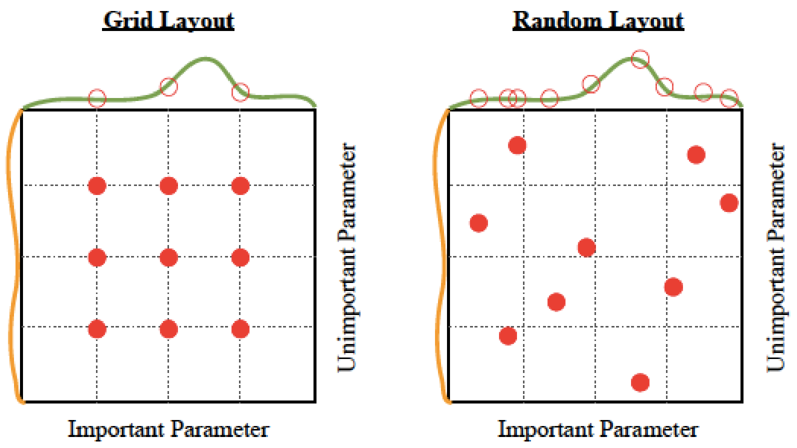
## Hyperparameter Optimization (HPO)

Hyperparameter Optimization (HPO) is the process of systematically searching for the best set of hyperparameters for a machine learning model to maximize its performance on a given task. Hyperparameters are model parameters that are set before training begins and cannot be learned directly from the data (e.g., learning rate, batch size, number of layers, or regularization strength). Properly tuned hyperparameters significantly affect the model's accuracy, efficiency, and generalization ability.

In AutoML, HPO is integrated into a broader pipeline that often includes model selection, data preprocessing, and feature engineering. This end-to-end automation simplifies the machine learning workflow for users by handling complex tasks with minimal manual intervention.

Common Methods for HPO include the following:

1. **Grid Search:** Grid search systematically evaluates all possible combinations of hyperparameters within a predefined grid. It is simple to implement and provides exhaustive coverage of the search space, ensuring no configuration is overlooked. However, it becomes computationally expensive and inefficient as the number of hyperparameters and their possible values increases, making it impractical for large or complex search spaces.
2. **Random Search:** Random search addresses some of the inefficiencies of grid search by randomly sampling hyperparameter combinations from a predefined range. This approach is often faster and can find good hyperparameter values efficiently by covering the search space more flexibly. However, it still requires significant computational resources for large search spaces and may not guarantee exhaustive exploration.



*Grid Search vs. Random Search.*

3. **Bayesian Optimization:** Bayesian optimization uses a **probabilistic surrogate model**, such as Gaussian Processes, to model the objective function and iteratively select promising hyperparameters. It balances exploration (trying new configurations) and exploitation (refining good ones), reducing the number of evaluations required. Despite its efficiency, Bayesian optimization introduces computational overhead in maintaining and updating the surrogate model.
4. **Hyperband:** Hyperband combines random search with early stopping to optimize resource allocation. It identifies poor-performing hyperparameter configurations early in the process, freeing resources to focus on more promising candidates. This method is highly scalable and efficient but depends on the feasibility of early stopping, which may not be suitable for all types of models or tasks.
5. **Evolutionary Algorithms:** Inspired by biological evolution, evolutionary algorithms optimize hyperparameters by simulating processes such as mutation, crossover, and selection. This approach is particularly effective for large and complex search spaces, as it can explore diverse configurations over generations. However, it is computationally intensive and tends to be slower compared to other methods.
6. **Gradient-Based Optimization:** Gradient-based optimization utilizes gradients of hyperparameters with respect to the validation loss to optimize them directly. It is particularly efficient for optimizing differentiable hyperparameters, such as learning rates in neural networks. However, this method is limited in applicability, as it cannot handle non-differentiable hyperparameters or categorical values.
7. **Reinforcement Learning (RL):** Reinforcement learning treats hyperparameter optimization as a sequential decision-making problem, where each action corresponds to choosing a set of hyperparameters. Rewards are based on validation performance, and the process evolves over multiple iterations. While effective for multi-step problems, RL is computationally expensive and can be complex to implement.

## Meta-learning

Meta-learning, often referred to as 'learning to learn,' is a concept in AutoML that leverages knowledge from previous machine learning tasks or trials to improve the efficiency and performance of future ones. In the context of AutoML, meta-learning enables the system to optimize processes like model selection, hyperparameter optimization (HPO), and pipeline design by utilizing patterns and insights from prior experiments or datasets. Meta-Learning Works in the following way:

1. **Learning from Historical Data**: AutoML systems build a "meta-dataset" by storing information from past tasks, such as dataset characteristics, model performance metrics, and hyperparameter outcomes. This enables the system to predict suitable models, hyperparameters, or workflows for new datasets.
2. **Building Meta-Models**: Meta-models are trained to map dataset characteristics (meta-features) to optimal ML pipelines or configurations. For instance, if high-dimensional datasets previously performed well with a specific algorithm, the system prioritizes that algorithm for similar datasets.
3. **Cold Start and Warm Start**:
   - **Cold Start**: Improves performance for new datasets without prior knowledge by leveraging general trends from related tasks.
   - **Warm Start**: Accelerates optimization for similar tasks by reusing previously successful configurations or models.

## Pipeline Automation

AutoML relies on algorithms for automated feature engineering, hyperparameter optimization (HPO), and model selection to streamline the machine learning process. However, to ensure these components work together as part of an automated pipeline, AutoML also incorporates additional concepts that enable pipeline automation. Some key concepts are explained below:

1. **Modularity:** The workflow is divided into distinct, independent stages (e.g., data preprocessing, feature engineering, model selection). Each stage performs a specific task, and the outputs from one stage are passed as inputs to the next. Modular pipelines offer flexibility, allowing individual components to be swapped or fine-tuned without disrupting the overall workflow.
2. **Workflow Generalization:** ML pipelines are designed to work across a variety of datasets and tasks with minimal customization. This is achieved through two key mechanisms:
   a) **Parameterization**: Configurable parameters allow pipelines to adapt to different inputs, such as specifying scaling techniques or model types.
   b) **Template Pipelines**: Predefined workflows (e.g., for classification or regression) provide a starting point that users can customize further.
3. **Pipeline Orchestration:** Pipeline orchestration is a critical layer at the core of AutoML, ensuring that individual components (e.g., feature engineering, HPO, model selection) and underlying concepts (e.g., modularity, abstraction, execution control) are coordinated effectively. It connects and manages the

execution of the entire workflow, handling dependencies, task scheduling, and resource optimization to ensure a seamless end-to-end process.

## Overview of Some Famous AutoML Services

### Auto-sklearn

Auto-sklearn automates the process of selecting and tuning machine learning models using scikit-learn's library of algorithms. It excels in structured datasets and provides robust solutions for supervised learning tasks like classification and regression. Its strength lies in its built-in ensemble learning capabilities and use of Bayesian optimization to identify the best models.

### TPOT

TPOT uses genetic algorithms to optimize machine learning pipelines, making it particularly useful for automating feature engineering and model selection. Its evolutionary approach iteratively improves pipeline performance, which is ideal for complex datasets where unconventional pipeline configurations may outperform standard methods.

### H2O AutoML

H2O AutoML offers a fully automated end-to-end machine learning pipeline that includes data preprocessing, model training, hyperparameter tuning, and model stacking. It's especially powerful for large-scale enterprise applications, thanks to its scalability and support for distributed computing, making it suitable for big data and cloud environments.

### PyCaret

PyCaret is a low-code AutoML library designed for Python, simplifying model development for beginners and rapid prototyping. It covers the entire machine learning pipeline, from data preprocessing to deployment, and is ideal for users looking for ease of use with minimal coding in scenarios like business analytics and small-scale projects.

### Google Cloud AutoML

Google Cloud AutoML provides scalable, cloud-based solutions for automating machine learning tasks. It supports diverse data types, including structured data, text, images, and video, making it highly suitable for businesses seeking customizable and production-ready models integrated with Google Cloud's infrastructure.