Machine Learning

|Hyperparameter Tuning and Model Selection

Prof. Dr. Mostafa Elhosseini

Professor of Smart Systems Engineering

https://youtube.com/drmelhosseini

Agenda

- Hyperparameter

Intro

- **Evaluating a model** is simple enough: just use a **test set**.
- But suppose you are hesitating between **two types of models** (say, a linear model and a polynomial model): how can you decide between them? One option is to **train both** and compare how well they generalize using the **test set**

Intro

- Now suppose that the linear model generalizes better, but you want to **apply some regularization to avoid overfitting**. The question is, how do you choose the value of the regularization hyperparameter? One option is to **train 100 different models using 100 different values for this hyperparameter**. Suppose you **find the best** hyperparameter value that produces a model with the lowest generalization error —say, just 5% error. You **launch this model into production**, but **unfortunately** it does not perform as well as expected and produces 15% errors. What just happened?

Development set – dev set

- The problem is that you **measured the generalization error multiple times on the test set**, and you adapted the model and hyperparameters to produce the best model for that particular set. This means the model is unlikely to perform as well on new data.
- A common solution to this problem is called **holdout validation**.
- you simply hold out **part of the training set** to evaluate several candidate models and select the best one. The new held-out set is called the validation set (or the **development set**, or **dev set**).

Development set – dev set

- **Data Splitting**: Divide your dataset into training, validation, and test sets.
- **Model Training**: Train multiple models with different hyperparameters on the reduced training set (excluding validation data).
- **Validation Evaluation**: Evaluate each model on the validation set and select the best-performing model.
- **Re-train Best Model**: Train the selected model on the full training set (including validation data).
- **Final Evaluation**: Assess the final model on the test set to estimate its generalization error.

Development set – dev set

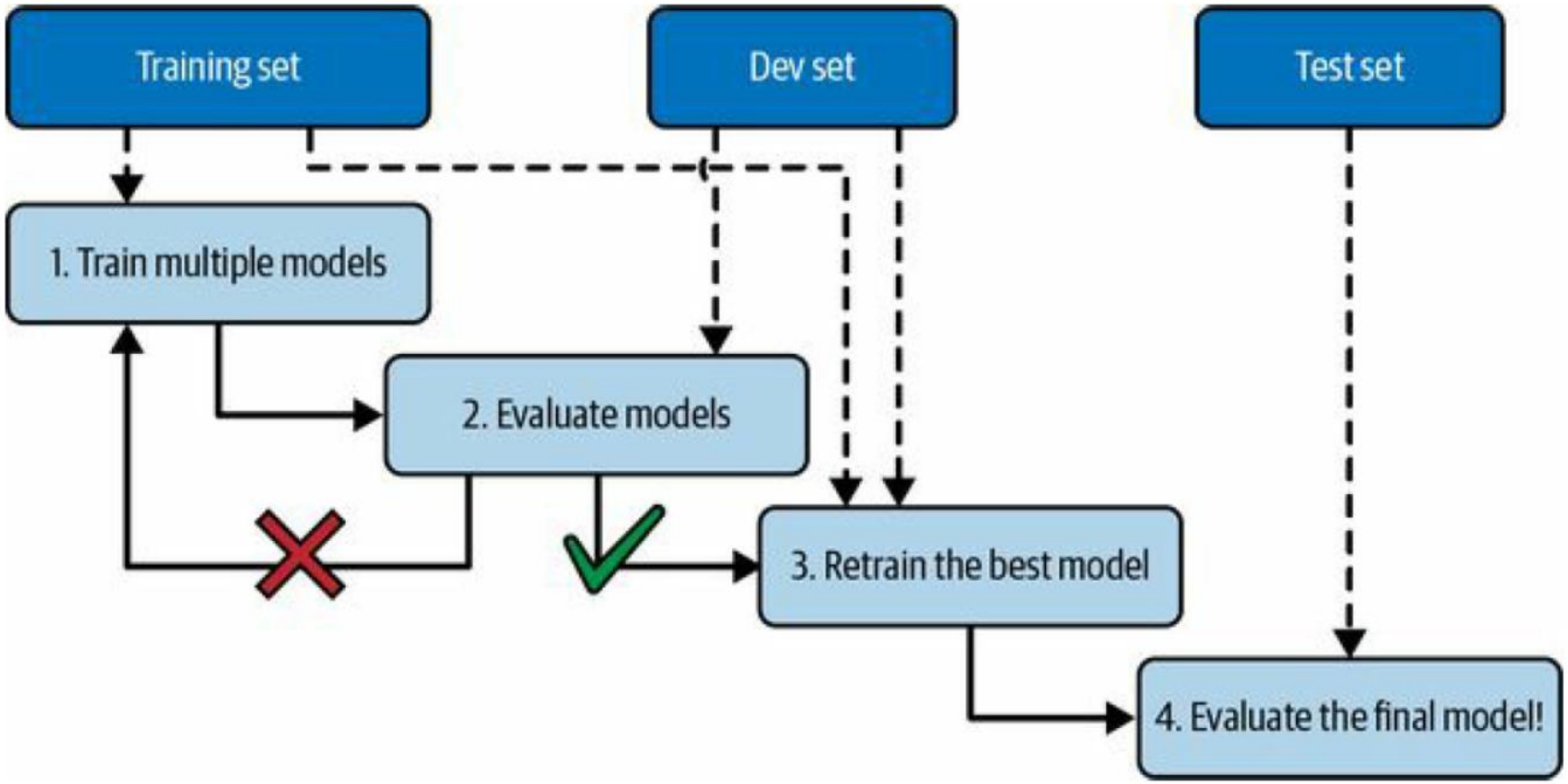


Figure 1-25. Model selection using holdout validation

Development set – dev set

- Data Splitting: Divide your dataset into training, validation, and test sets.
- Model Training: Train multiple models with different hyperparameters on the reduced training set (excluding validation data).
- Validation Evaluation: Evaluate each model on the validation set and select the best-performing model.
- Re-train Best Model: Train the selected model on the full training set (including validation data).
- Final Evaluation: Assess the final model on the test set to estimate its generalization error.

Development set – dev set

Development set used to:

- **Model Selection**: Evaluate and compare different models or algorithms on this set to select the best performer for the task.
- **Avoiding Overfitting**: Using a separate set for development helps in preventing the model from overfitting to the training data.
- **Performance Evaluation**: While the test set is used for the final evaluation of the model, the development set provides an ongoing, less biased evaluation of the model's performance during the development phase. This continuous feedback is essential for iterative improvement.

Development set – dev set

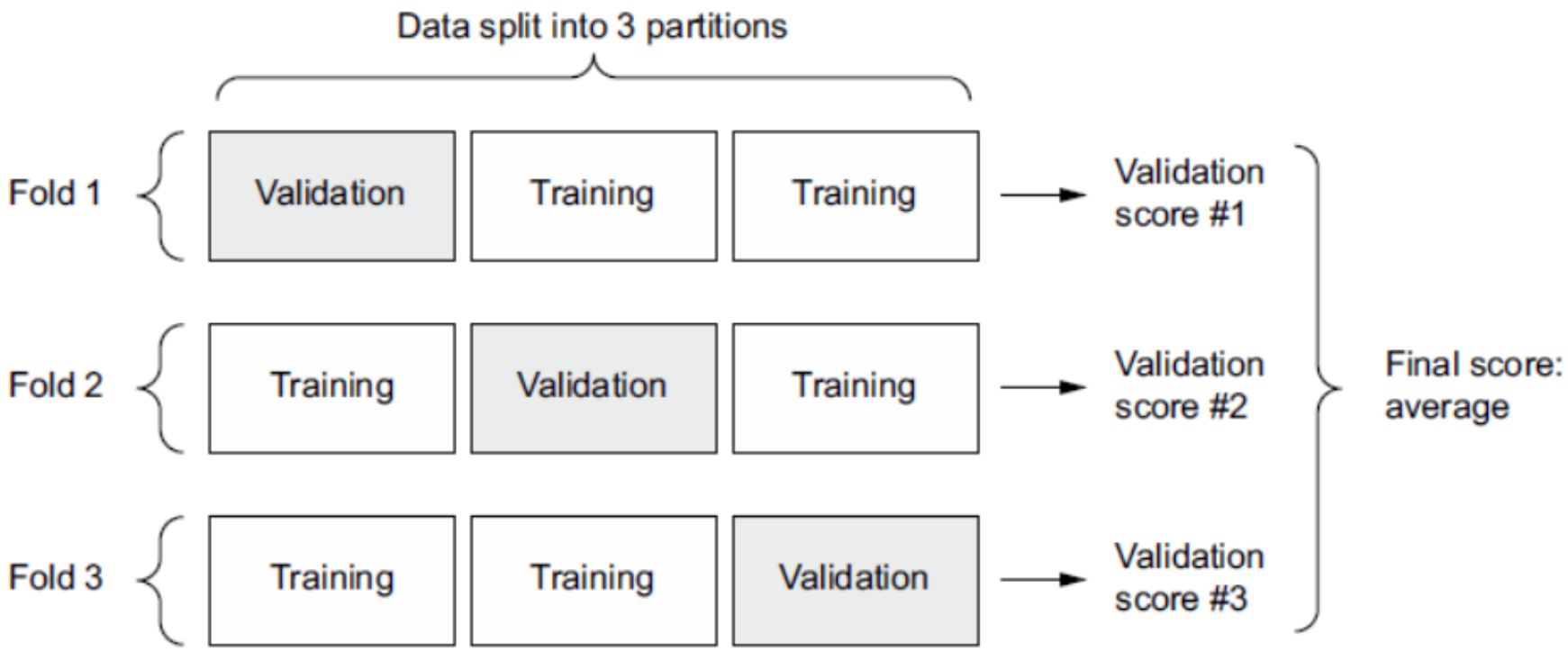
Development set used to:

- **Data Splitting**: In a typical ML project, data is divided into three parts: the training set, the development/validation set, and the test set. The training set is used to train the model, the development set to validate and improve it, and the test set to evaluate its final performance.
- **Early Stopping**: The development set can be used to implement early stopping, which is a form of regularization used to avoid overfitting. If the performance on the development set begins to degrade (while performance on the training set continues to improve), it's a sign that the model is starting to overfit, and training can be stopped.

Cross-validation

- Cross-validation is a statistical method used in machine learning to evaluate the generalizability of a model's predictions.
- It involves partitioning the original dataset into a training set to train the model and a validation (or test) set to evaluate it.
- The key feature of cross-validation is that it is repeated several times, with different partitions, to reduce variability in the assessment of the model's performance

Cross-validation



Cross-validation

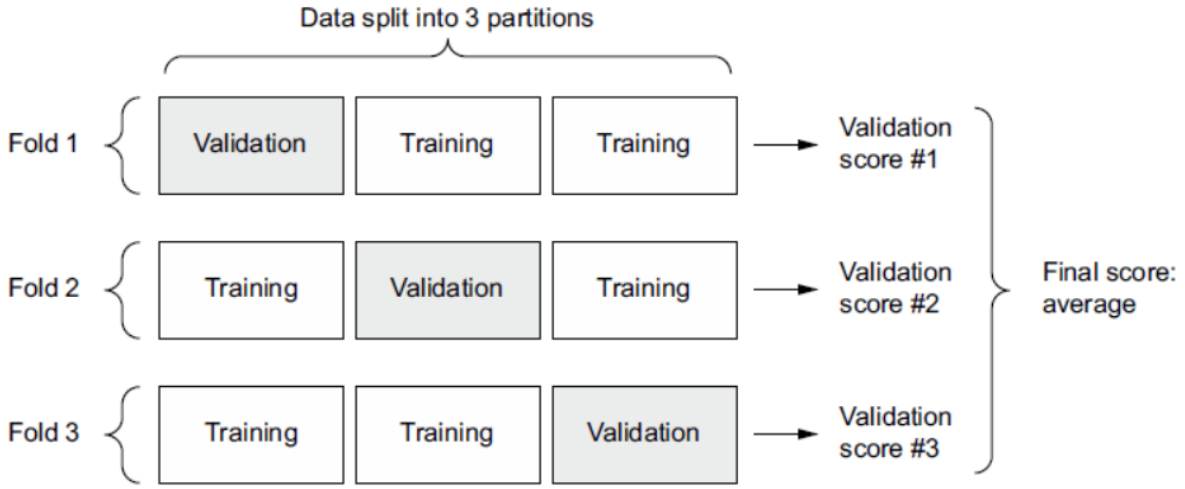
Steps:

- Splitting: Divide the dataset into 'k' equal subsets or folds.
- Model Training: For each unique group, take the group as a hold-out or test data set, and take the remaining groups as a training data set.
- Validation: Fit a model on the training set and evaluate it on the test set.
- Repeat: Repeat this process 'k' times, each time with a different fold designated as the test set.
- Aggregation: Aggregate the results from each fold to produce a single estimation.

Cross-validation

Types:

- **K-Fold Cross-Validation**: The data is divided into 'k' subsets, and the holdout method is repeated 'k' times. Each time, one of the 'k' subsets is used as the test set, and the other 'k-1' subsets are put together to form a training set.



Cross-validation

Types:

- **Stratified K-Fold Cross-Validation**: Similar to K-Fold, but in this variant, the folds are made by preserving the percentage of samples for each class, which is particularly useful for imbalanced datasets.

Cross-validation

Types:

- **Leave-One-Out Cross-Validation (LOOCV)**: A special case of k-fold cross-validation where 'k' equals the number of data points in the dataset. This means that for each iteration, one data point is used as the test set and the rest as the training set. This method is **computationally expensive** but can provide a thorough assessment for small datasets..

Data Mismatch

- **Data mismatch** refers to a situation where there's a significant difference between the training data and the data the model encounters in real-world applications, or between the training data and the validation/test data.
- both the validation set and the test set must be as representative as possible of the data you expect to use in production
- This discrepancy can lead to a model that performs well on training and validation datasets but poorly on real-world or test data

Data Mismatch

Data mismatch can occur due to various factors:

- **Feature Distribution**: The features or characteristics of the data used in training may not adequately represent those found in real-world or test data. For example, a **model trained on high-resolution images** might not perform well on lower-resolution images in production.

Data Mismatch

Data mismatch can occur due to various factors:

- **Data Source Variation**: The training data might come from a different source than the real-world or test data, leading to variations that the model hasn't learned to handle. For example, a **speech recognition** system trained on **studio-quality audio** recordings might struggle with noisy, real-world audio clips.

Data Mismatch

Data mismatch can occur due to various factors:

- **Temporal Changes**: The data's nature might change over time, a phenomenon known as concept drift. A model trained on past data may not perform well on current data if the underlying data distribution has changed.

Data Mismatch

Data mismatch can occur due to various factors:

- **Label Distribution**: The distribution of labels in the training set might not match the distribution in the real-world or test data, leading to biased predictions.
- Label distribution mismatch refers to the situation where the proportions of different categories (labels) in your training data do not reflect those in the real-world or test data. For example, if you're training a model to classify images as either cats or dogs and **90% of your training images are of cats, the model might become biased towards predicting 'cat' more often**. This is because it 'learns' from the training data that most images are supposed to be cats. If the real-world data you eventually apply the model to has a more balanced mix of cats and dogs, the model might still predict 'cat' too frequently, leading to inaccurate or biased predictions

Data Mismatch

Data mismatch can occur due to various factors:

- **Synthetic Data**: Models trained on synthetic or augmented data might face difficulties when applied to real data due to the inherent differences between synthetic and natural datasets.

Data Mismatch

Data mismatch can occur due to various factors:

- **Synthetic Data**: Models trained on synthetic or augmented data might face difficulties when applied to real data due to the inherent differences between synthetic and natural datasets.

Data Mismatch

- Large training datasets are often readily available but may not align with real-world production data.
- Example: A **mobile app** for identifying flower species can use millions of web-sourced images for training.
- However, these images may not accurately represent those captured by users through the app.

Data Mismatch

- When dealing with data that isn't perfectly representative of production scenarios, like the flower identification app example, it's crucial to ensure that both the validation and test sets closely mirror the actual data you expect in production. This means using only the representative images for these sets

Data Mismatch

Here's a simplified breakdown

- **Validation and Test Sets**: These should only contain representative images, akin to those captured by the app in real-world use.
- **Data Splitting**: Shuffle the representative images, allocating half to the validation set and half to the test set, while avoiding any duplicates in both sets.
- **Training on Non-representative Data**: If you train your model on easily available but non-representative data (like web images), and then it performs poorly on the validation set, it could be due to two reasons:
  - **Overfitting**: The model might have learned the training data too well, capturing noise and specifics not applicable to the representative data.
  - **Data Mismatch**: The discrepancy between the training data (web images) and the real-world app data might be causing the poor performance.

Data Mismatch - Solution

- **Train-Dev Set**: Reserve a portion of web-sourced training images as a "**train-dev**" set, as introduced by Andrew Ng.
- **Initial Evaluation**: Train the model on the main training set, then test on the train-dev set.
  - **Poor Performance on Train-Dev**: Indicates **overfitting**. Solutions include simplifying the model, adding regularization, acquiring more data, or cleaning the current dataset.
  - **Good Performance on Train-Dev**: Proceed to evaluate on the development (dev) set.

Data Mismatch - Solution

- **Dev Set Evaluation**:
  - Poor Performance on Dev Set: Suggests a data **mismatch issue**. Address this by preprocessing web images to more closely resemble app-captured images and retrain.
- **Final Steps**: Once the model shows satisfactory results on both train-dev and dev sets, conduct a final evaluation on the test set to gauge expected real-world

Data Mismatch - Solution

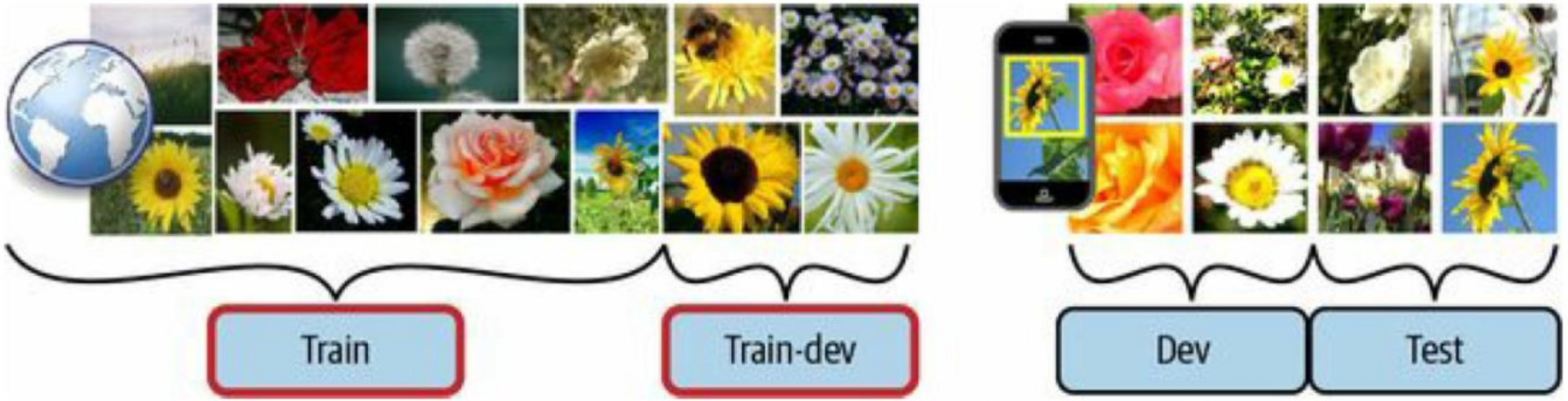


Figure 1-26. When real data is scarce (right), you may use similar abundant data (left) for training and hold out some of it in a train-dev set to evaluate overfitting; the real data is then used to evaluate data mismatch (dev set) and to evaluate the final model's performance (test set)

No Free Lunch Theorem

- Model Simplification: Selecting a model involves simplifying data representations to discard irrelevant details, assuming these simplifications won't affect generalization to new instances.
- Implicit Assumptions: Choosing a model type (e.g., linear model) implies assumptions about data characteristics (e.g., data's linearity).

No Free Lunch Theorem

- For instance, if you're analyzing the relationship between the number of hours studied and exam scores, choosing a linear regression model implies the assumption that there's a straight-line (linear) relationship between these variables.
- Essentially, you're assuming that as study hours increase, exam scores will increase in a consistent, predictable manner, following a linear trend. This model would not account for complexities like plateaus in score improvement after certain study durations, which might better fit a non-linear model.

No Free Lunch Theorem

- The No Free Lunch (NFL) theorem, formulated by David Wolpert and William Macready in the mid-1990s, is a principle in the field of optimization and machine learning. It states that no single optimization algorithm (or machine learning model) is universally superior to all others for all types of problems. The theorem asserts that when averaged across all possible problems, every algorithm performs equally well, given certain mathematical conditions and definitions of performance and problems.

No Free Lunch Theorem

- In the context of machine learning, this implies that there is no one-size-fits-all model that will always perform best for every type of data or task. The effectiveness of a model depends on how well its assumptions and structure match the specific characteristics of the problem at hand. Therefore, the choice of model and algorithm must be informed by the nature of the data and the specific requirements of the task, necessitating a process of experimentation and model comparison to identify the most suitable approach for a given situation.
- No Free Lunch Theorem: David Wolpert's 1996 paper established that without specific data assumptions, no single model can be preferred over others. Model effectiveness varies by dataset.

No Free Lunch Theorem

In brief:

- No Universal Superiority: No single algorithm or model outperforms all others across all problem types.
- Average Performance Equality: Across all possible problems, every algorithm's performance averages out to be equal.
- Model Effectiveness: Depends on the match between model assumptions/structure and problem characteristics.
- Task-Specific Model Selection: Choosing the right model requires considering the data's nature and the task's specific needs.

No Free Lunch Theorem

- Practical Approach: Due to the impracticality of evaluating all possible models, assumptions are made about the data to narrow down the choice to a few reasonable models based on the task's complexity.