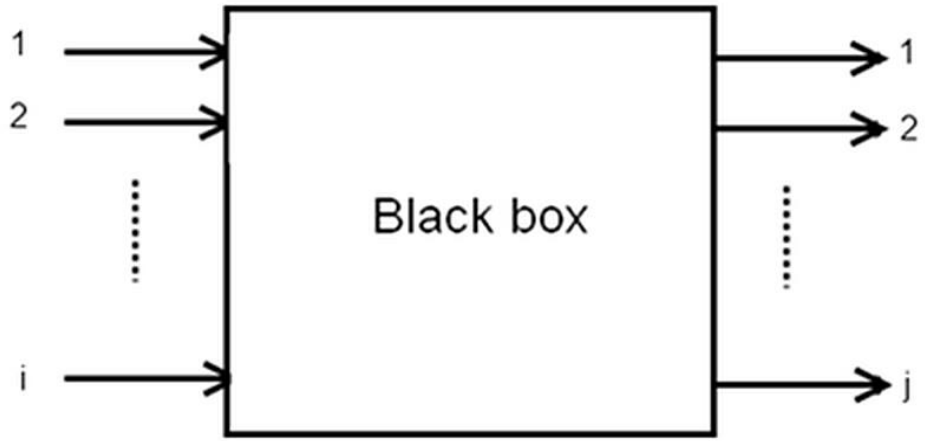
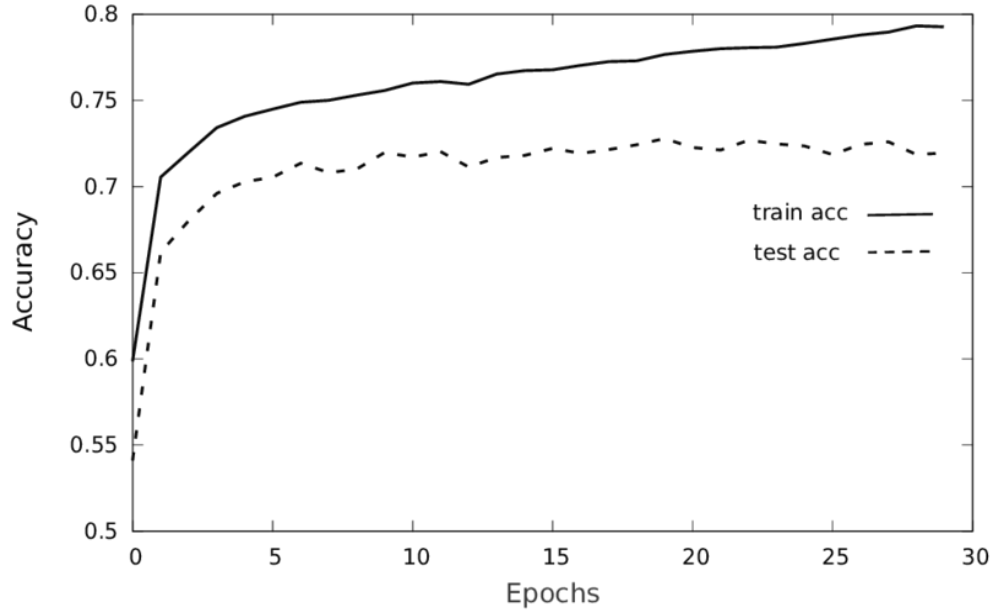
Hyperparameter tuning of metamodels

Week 11 - Advanced Topic 5

- **Metamodel** is a simplified model of an actual model, to make fast calculation of the properties of the actual model





- **Hyperparameter** is a parameter that is set to adjust the learning process of a model, before training

How do we tune hyperparameters?

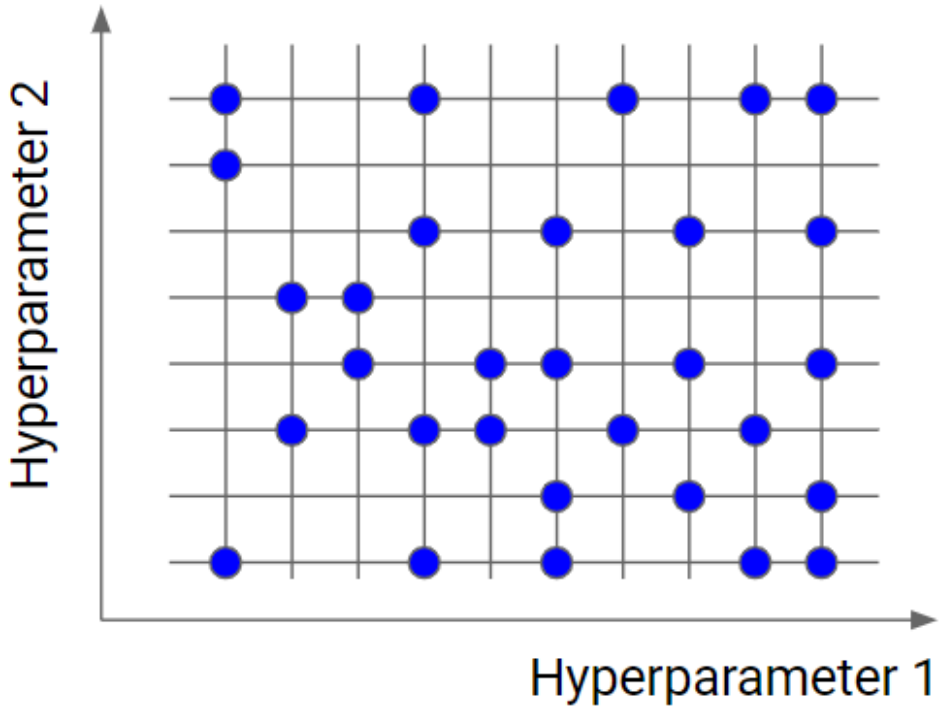
Image 1: https://www.researchgate.net/figure/nputs-and-outputs-of-the-black-box-model_fig2_268461818
Image 2: https://www.researchgate.net/figure/Convergence-of-the-training-and-test-accuracy-as-a-function-of-the-number-of-epochs_fig3_318771258

Types of metamodels

- **Polynomial Equations**: Used in mathematical modeling, these metamodels represent complex relationships using polynomial equations, making it easier to analyze and predict behaviors.
- **Artificial Neural Networks**: These metamodels are used in machine learning to model complex patterns and relationships in data. They are particularly useful for tasks like classification and prediction.
- **Ontology Metamodels**: Converting human language into machine language

Tuning hyperparameters

- **Manual Search**: Sometimes, domain knowledge can guide the selection of hyperparameters. This involves manually adjusting hyperparameters based on experience and intuition.
- **Grid Search**: This method involves systematically searching through a predefined set of hyperparameter values. It evaluates the model for each combination of hyperparameters and selects the best one based on performance metrics.
- **Random Search**: Instead of searching through all possible combinations, random search samples hyperparameter values from a specified distribution. This can be more efficient than grid search, especially when dealing with a large number of hyperparameters.
- **Bayesian Optimization**: This technique uses a probabilistic model to predict the performance of different hyperparameter settings. It then selects the most promising hyperparameters to evaluate, balancing exploration and exploitation
- **Optuna**: Works through a process called sequential model-based optimization (SMBO), which iteratively refines the search for optimal hyperparameters using a probabilistic model



Motivation

- Enhance computational efficiency
- Balance bias and variance
- Optimize model robustness
- Transferability to new data
- Ensure stability in iterative processes
- Improve predictive accuracy

Code example: Random Forest Regressor Model

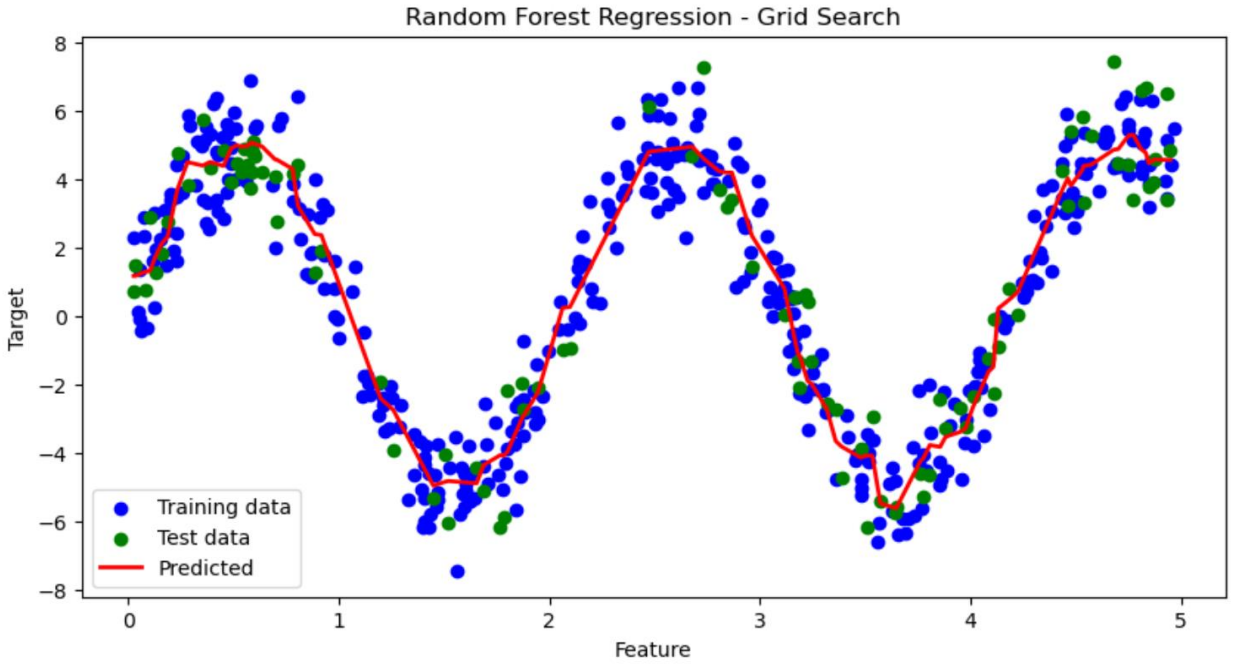
- **Grid search**: picks from discretized space and ends with the best configuration.

- **Randomized search**: Randomly picks from parameter space and ends with a stopping criteria.

  e.g. Bayesian Optimization

- **Automated Tuning**: Finds optimal parameters by utilizing different methods (grid search, random search, etc.)
- how does it work
- **Optuna**: Works through a process called **sequential model-based optimization** (SMBO), which iteratively refines the search for optimal hyperparameters using a probabilistic model.

Grid Search

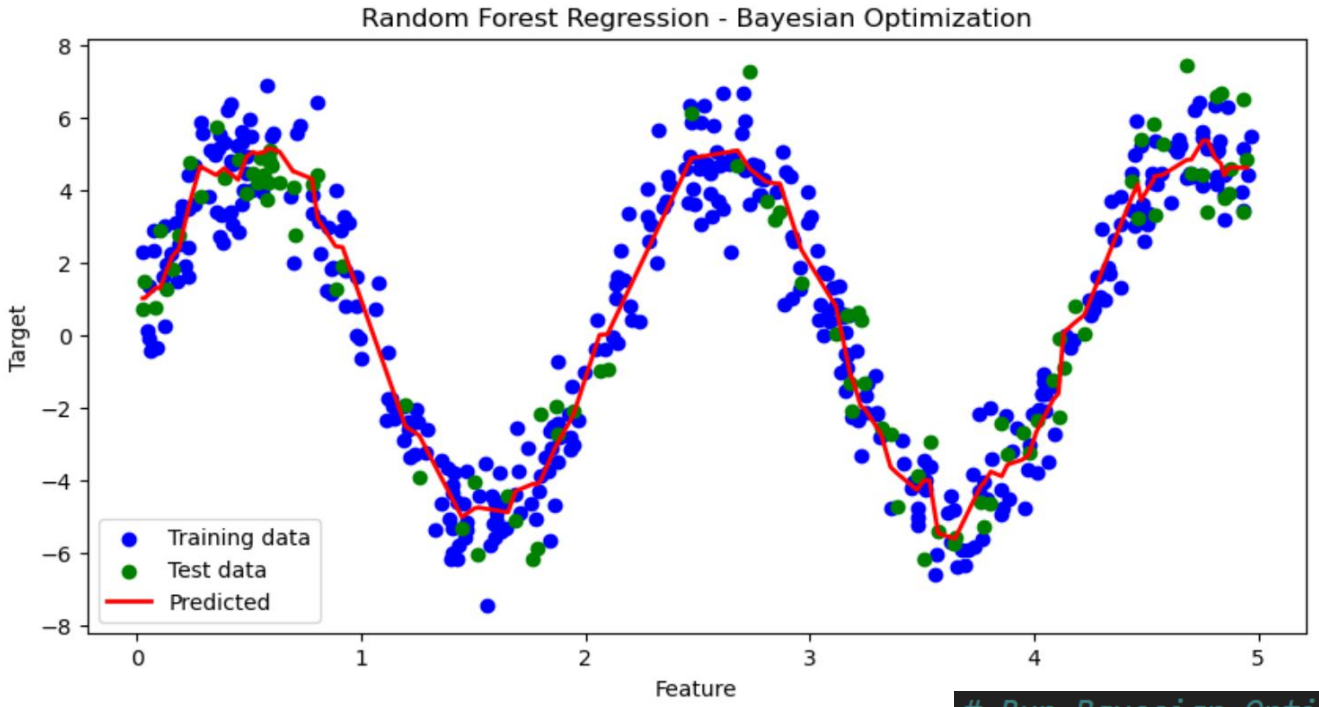



```
# Get the best estimator after Grid Search
best_rf_grid = grid_search.best_estimator_
```

```
# Define a small hyperparameter grid for tuning
hyperparam_grid = {
    "n_estimators": np.linspace(10, 200, num=5, dtype=int),  #
    "max_depth": np.linspace(5, 30, num=3, dtype=int),       #
    "min_samples_split": np.linspace(2, 20, num=3, dtype=int),
    "min_samples_leaf": np.linspace(1, 10, num=3, dtype=int)
```

```
Best model found by Grid Search:
RandomForestRegressor(max_depth=17, min_samples_leaf=5, n_estimators=200,
                      random_state=42)
```

Randomized Search (Bayesian Optimization)



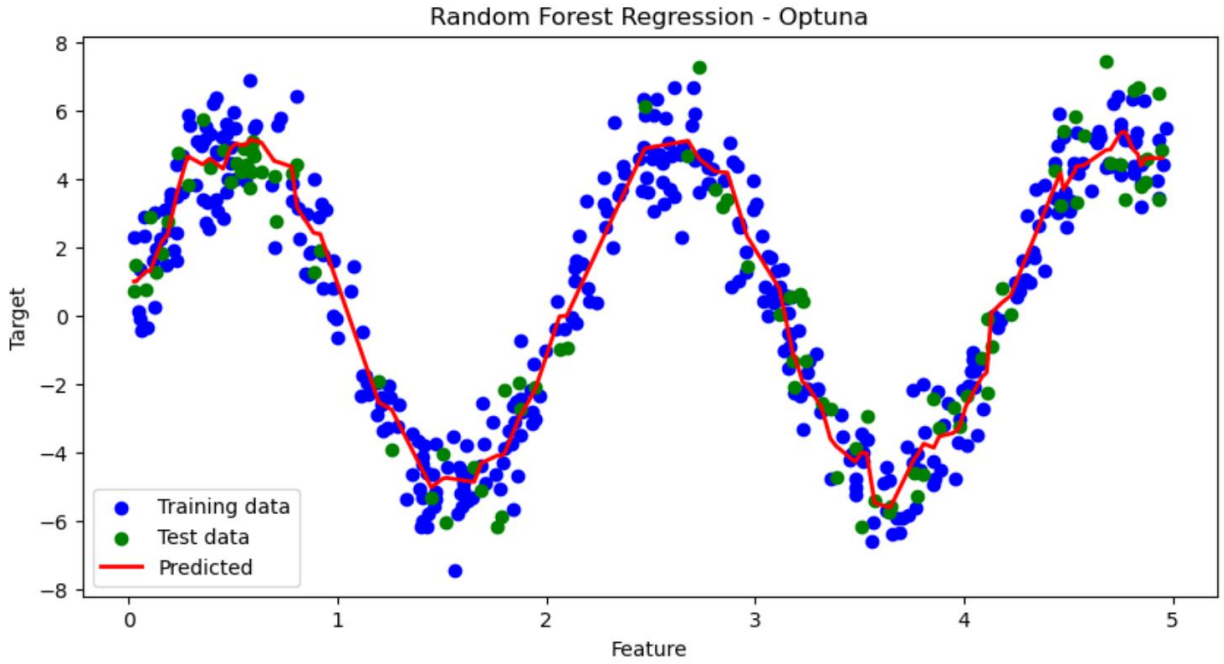

```
pip install bayesian-optimization
```

```
# Define the parameter space
param_bounds = {
    'n_estimators': (50, 200),
    'max_depth': (5, 30),
    'min_samples_split': (2, 20),
    'min_samples_leaf': (1, 10)
```

```
# Run Bayesian Optimization
optimizer = BayesianOptimization(f=rf_cv, pbounds=param_bounds, random_state=42)
optimizer.maximize(init_points=10, n_iter=30)
```

```
Best hyperparameters found:
{'max_depth': 17, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 199}
```

Automated Tuning: Optuna

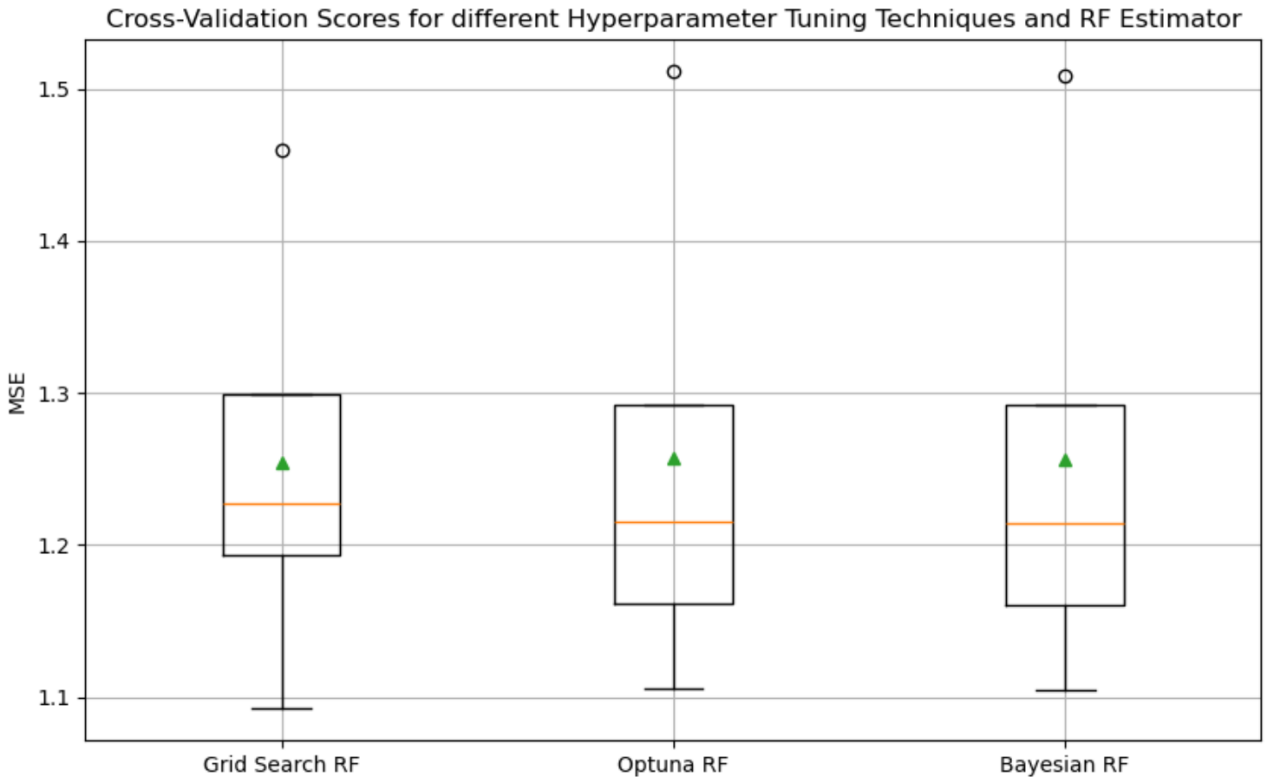


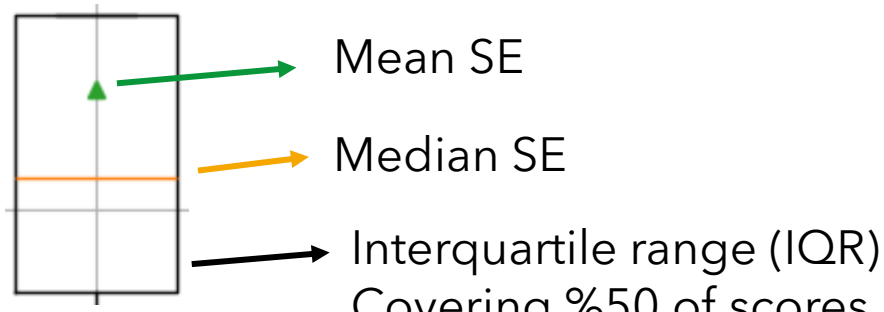

```python
# Suggest hyperparameters
n_estimators = trial.suggest_int("n_estimators", 10, 200)
max_depth = trial.suggest_int("max_depth", 5, 30)
min_samples_split = trial.suggest_int("min_samples_split", 2, 20)
min_samples_leaf = trial.suggest_int("min_samples_leaf", 1, 10)
```

```python
# Run Optuna for hyperparameter tuning
study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=50)
```

```
Best hyperparameters found by Optuna:
{'n_estimators': 142, 'max_depth': 14, 'min_samples_split': 2, 'min_samples_leaf': 4}
```

Cross-validation Analysis

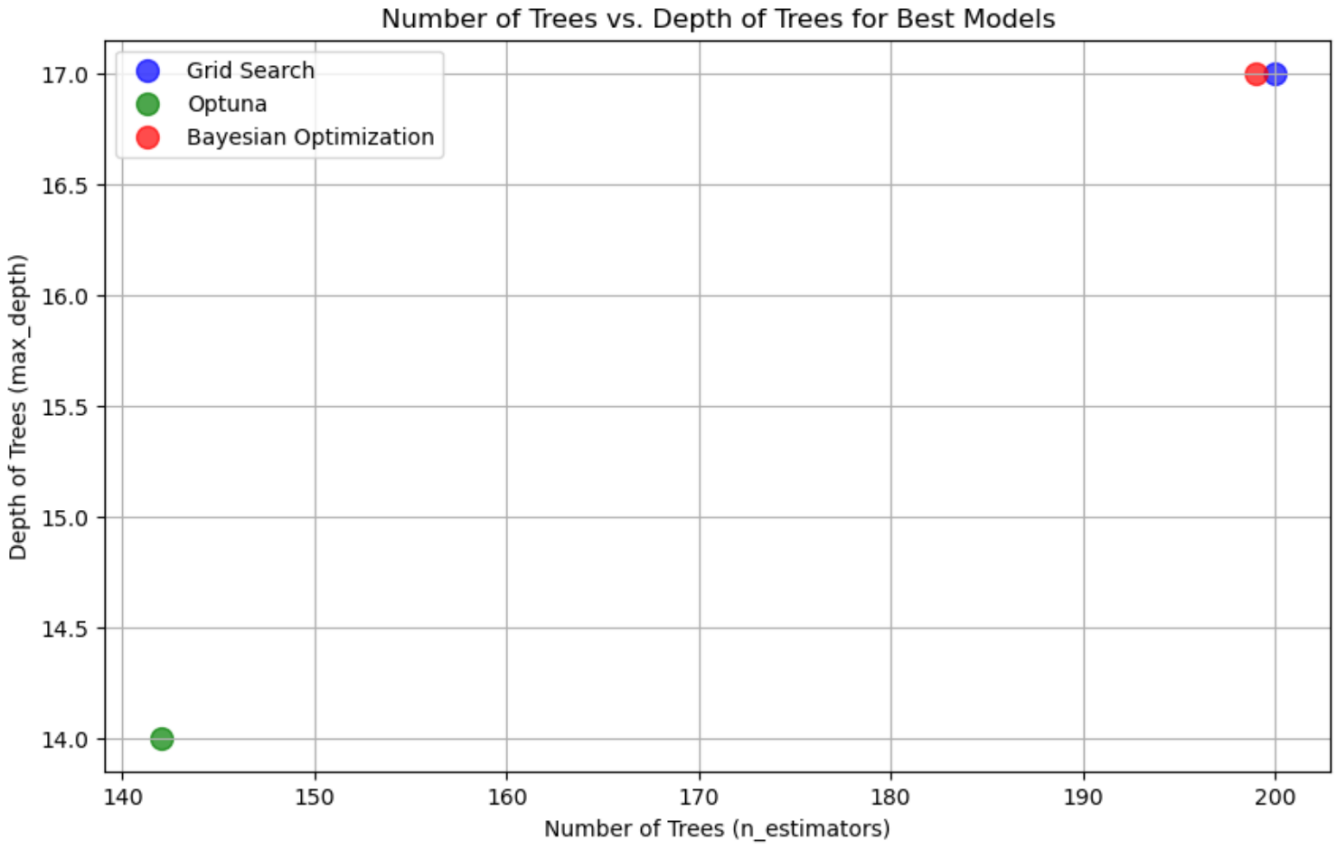





```
Grid Search RF: Mean MSE = 1.2545, Std = 0.1224
Optuna RF: Mean MSE = 1.2573, Std = 0.1411
Bayesian RF: Mean MSE = 1.2561, Std = 0.1406
```

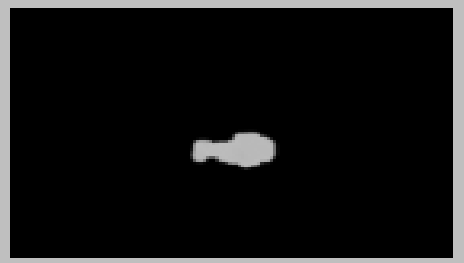
- Grid : 54.26 s
- Optuna : 26.33 s
- Bayesian : 41.22 s

Complexity and Performance Trade-Off

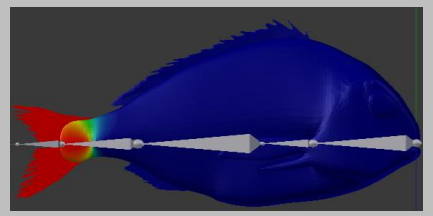


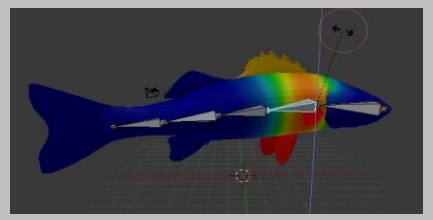
Particle Swarm Optimization (PSO)

Image acquisition
-
Image processing

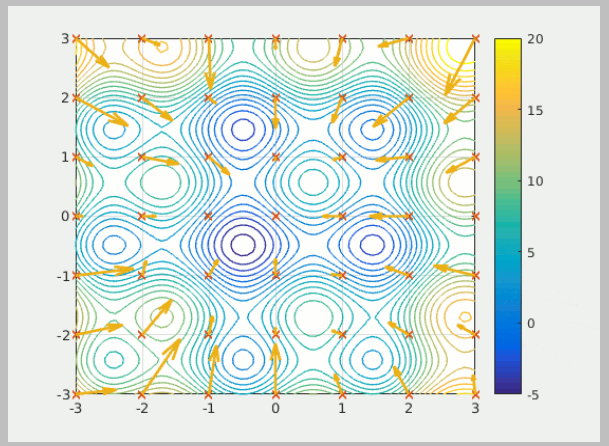


3D model





PSO



Results

Conclusions

- Reduce computational costs
- Higher robustness and stability
- Greater efficiency in convergence of iterative models
- Better predictive quality
- Optimal trade-off between simplicity and complexity

Sources

[1] Abel AK, October 2023, *A guide to hyperparameter tuning: Enhancing machine learning models. Online:* https://medium.com/@abelkuriakose/a-guide-to-hyperparameter-tuning-enhancing-machine-learning-models-69dc9e0f02ea

[2] Russell R. Barton, 2020, *TUTORIAL: METAMODELING FOR SIMULATION*, Proceedings of the 2020 Winter Simulation Conference, https://ieeexplore.ieee.org/document/9384059

[3] Mantovani, Rafael G., et al. "Hyper-parameter tuning of a decision tree induction algorithm." *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, 2016.

Thank you!