Task 4: Hyperparameter Optimization

Bayesian Optimization and Optuna Strategies

Luisa Faust

2025

1 Objective

The goal of this exercise was to design a hyperparameter optimization pipeline that combines a **custom Bayesian optimizer** based on a *Gaussian Process (GP)* with several optimization strategies provided by **Optuna**, including the Tree-structured Parzen Estimator (TPE), the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), and Random Search. The task builds directly on the dynamically constructed feature set from Task 3 and evaluates which optimization strategy achieves the best model performance and computational efficiency.

2 Background

2.1 Bayesian Optimization

Bayesian Optimization (BO) is an efficient method to optimize expensive, black-box objective functions. It models the objective function $f(x)$ using a surrogate model—in this case a **Gaussian Process**—which provides both a mean prediction $\mu(x)$ and an uncertainty estimate $\sigma(x)$. New evaluation points are chosen according to an **acquisition function** (e.g. Expected Improvement) that balances *exploration* (uncertain areas) and *exploitation* (promising areas).

2.2 Gaussian Process Regressor

A Gaussian Process is a non-parametric model defined by its mean and covariance function:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

It provides a probabilistic prediction of the objective value at unseen points and is well-suited for continuous optimization spaces.

2.3 Optuna Framework

Optuna (https://optuna.org/) is an automatic hyperparameter optimization library that implements several sampling strategies:

- **Random Sampling:** purely stochastic baseline.
- **TPE (Tree-structured Parzen Estimator):** models $p(x|y)$ and $p(y)$ to balance exploration/exploitation probabilistically.
- **CMA-ES (Covariance Matrix Adaptation Evolution Strategy):** an evolutionary algorithm that adapts its search distribution iteratively based on previous samples.

3 Implementation

The chosen model was a **Random Forest Classifier** trained on the dynamic feature set. The objective function returned the weighted F1-score of a 10-fold stratified cross-validation.

Optimized Hyperparameters:

Parameter	Range	Description
n_estimators	[50, 300]	Number of trees
max_depth	[3, 20]	Maximum tree depth
min_samples_split	[2, 15]	Min. samples to split a node
min_samples_leaf	[1, 10]	Min. samples per leaf node

Two optimization pipelines were implemented:

1. A **custom Bayesian optimizer** using `sklearn.gaussian_process.GaussianProcessRegressor`.
2. **Optuna-based studies** with 30 trials each for TPE, CMA-ES, and Random sampling.

4 Results and Discussion

4.1 Feature Extraction and Baseline

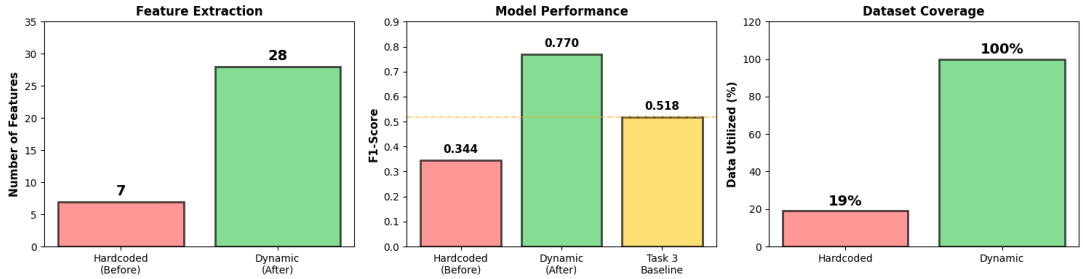

Figure 1: Feature extraction comparison: hardcoded vs. dynamic feature set.

Dynamic feature extraction increased the feature count from 7 to 28 and improved the F1-score from 0.344 to 0.770, significantly outperforming the Task 3 baseline (0.518). Dataset coverage rose from 19 % to 100 %, showing that the new feature pipeline leverages the entire dataset more effectively.

4.2 Strategy Comparison and Efficiency (All Optimizers)

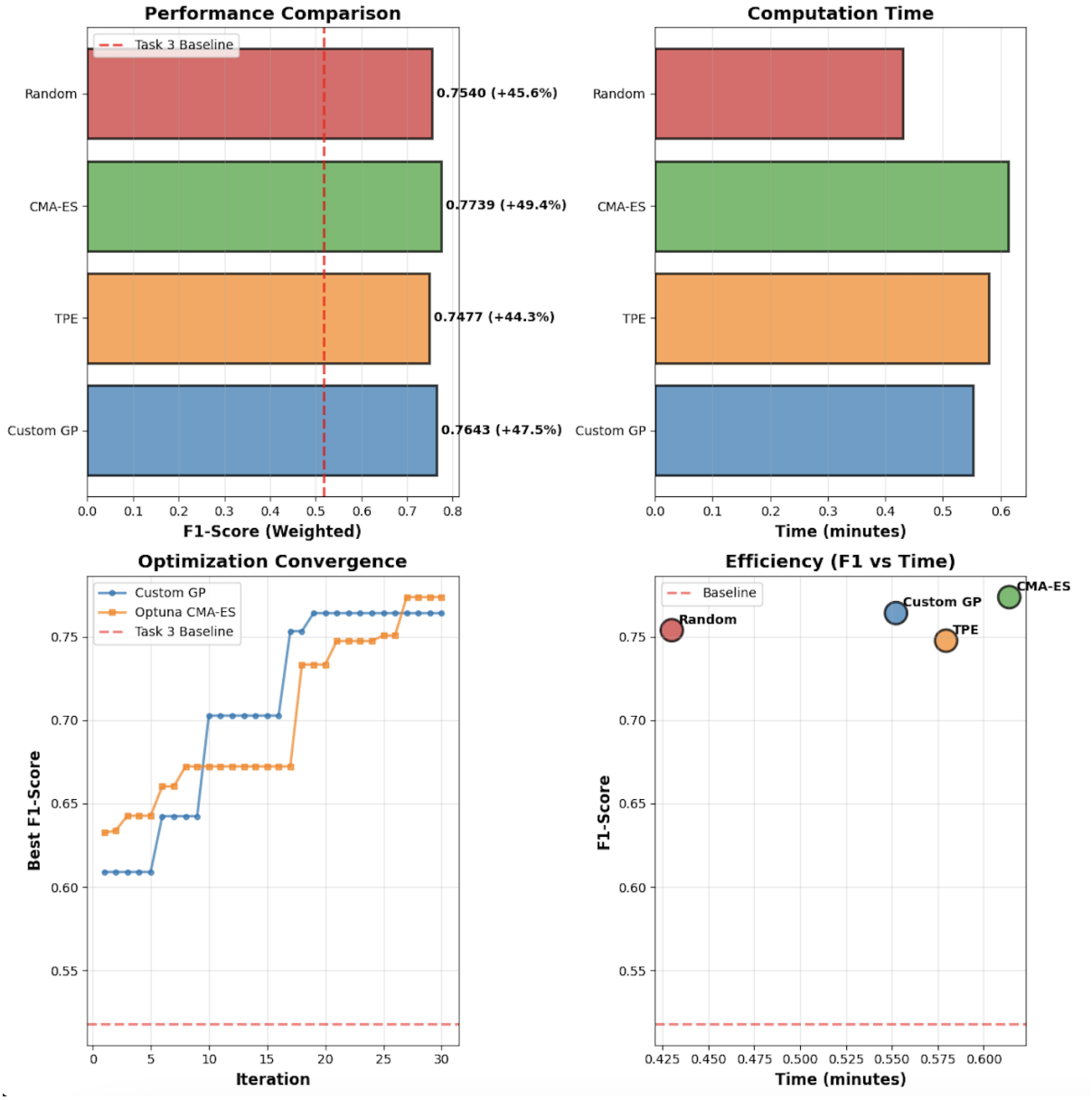

Figure 2: Strategy dashboard: best F1 (top-left), runtime (top-right), convergence curves (bottom-left), and efficiency (bottom-right) comparing CMA-ES, TPE, Random, and the custom GP.

How to read the dashboard.

- **Top-left (Best F1 per strategy):** Bars show the peak validation F1 each optimizer achieved within the same budget (number of trials). Higher is better.
- **Top-right (Runtime):** Total wall-clock time per study. Lower is better, but interpret together with F1.
- **Bottom-left (Convergence curves):** Best-so-far F1 vs. trials. Steeper early gains ⇒ faster exploitation; long plateaus ⇒ stalled search.
- **Bottom-right (Efficiency):** F1 as a function of time. Curves that rise quickly and plateau high are preferable.

Interpretation.

- **Winner: CMA-ES** reached the top F1 (≈ 0.774) within the budget and showed stable convergence (rapid early ascent, then plateau).
- **Custom GP (Bayesian)** closely trailed CMA-ES, confirming that the Gaussian-Process optimizer is correctly implemented and sample-efficient.
- **TPE** converged a bit slower in this run and plateaued slightly lower (≈ 0.748).
- **Random Search** was fastest in wall-clock time but plateaued earlier (lower asymptote).

Strategy	Best F1	Runtime [min]	Gain vs. baseline
CMA-ES	**0.7739**	0.6	+49.4 %
Custom GP (BO)	0.7643	0.6	+47.5 %
TPE	0.7477	0.6	+44.3 %
Random Search	0.7540	**0.4**	+45.6 %

4.3 Optuna Analyses (Ordered as in the Notebook)

1) Hyperparameter Importance

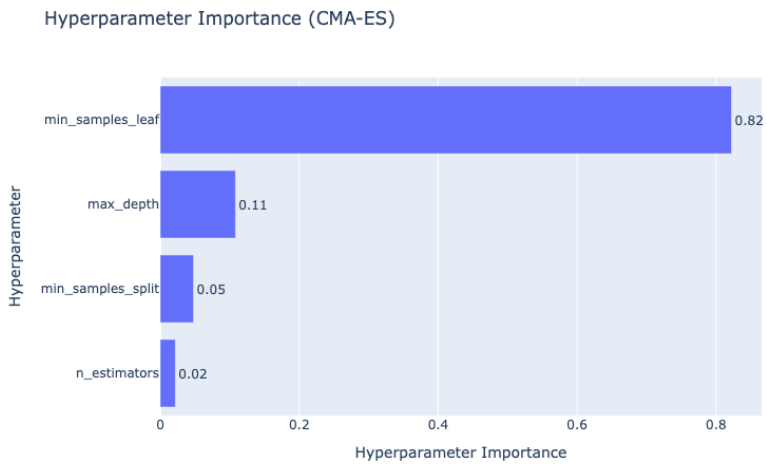

Figure 3: Hyperparameter importance (CMA-ES study). Bars approximate each parameter's contribution to F1 variance across trials.

Interpretation.

- `min_samples_leaf` dominates (≈ 0.82): critical for generalization. Too small $\Rightarrow$ overfit; too large $\Rightarrow$ underfit.
- `max_depth` contributes moderately (≈ 0.11): depth enables interaction modeling but with diminishing returns.
- `min_samples_split` and `n_estimators` have minor effects within the searched ranges.

2) Optimization History

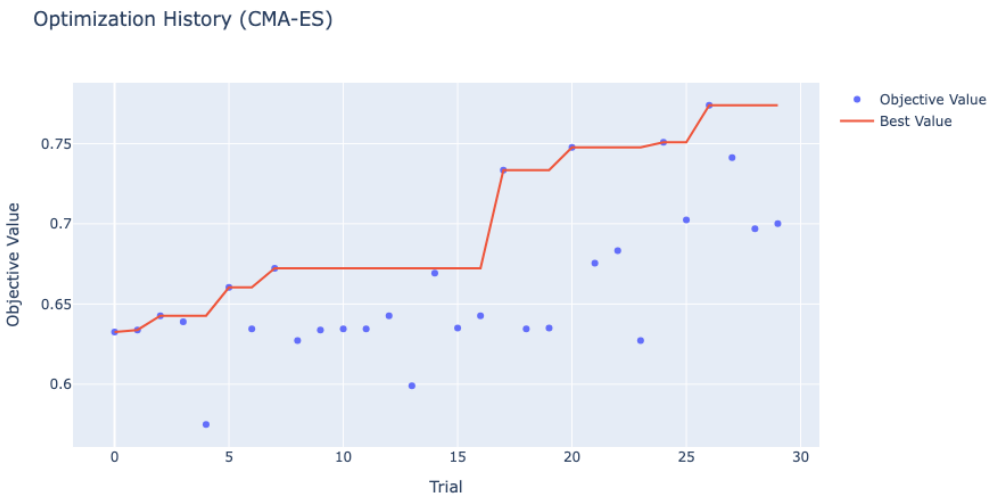

Figure 4: Optimization history (CMA-ES): blue dots are per-trial F1; the red curve is the running best.

Interpretation.

- Early exploration around F1 $\approx$ 0.62–0.66.
- Mid-run jumps to $\approx$ 0.74–0.75 (trials $\approx$ 17–21).
- Plateau near $\approx$ 0.78, indicating convergence to a high-performing region.

3) Parallel Coordinate Plot

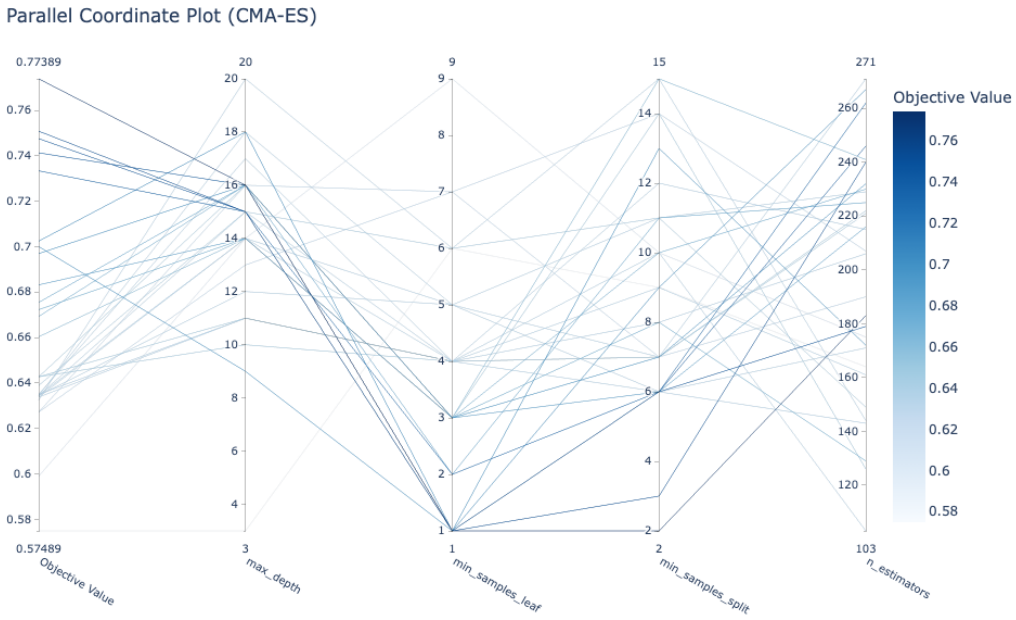

Figure 5: Parallel coordinates (CMA-ES): each polyline is a trial; axes are hyperparameters and the resulting F1. Darker lines $\Rightarrow$ higher F1.

Interpretation.

- High-F1 lines cluster at `min_samples_leaf` = 1–2.
- `max_depth` around 12–18 is consistently strong; too shallow or extremely deep underperform.
- `n_estimators` spreads across 100–270 with weak sensitivity, consistent with its low importance.

4.4 Conclusions

Part 1: Strategy Comparison

Across all four strategies, **CMA-ES** emerged as the most effective optimizer, combining fast convergence and high stability with the best final F1-score (≈ 0.774). The **custom Bayesian optimizer (Gaussian Process)** achieved nearly identical results, proving that a well-implemented GP can rival state-of-the-art libraries in sample efficiency. **TPE** performed competitively but required more trials to converge, while **Random Search** offered reasonable results at minimal runtime, serving as a strong baseline for smaller budgets. Overall, structured optimization, either evolutionary or Bayesian, clearly outperformed uninformed random sampling, particularly when meaningful, engineered features were provided.

Part 2: Optuna Analysis

The Optuna visualizations provided deep insight into how the optimizer explored the parameter space:

- **Hyperparameter importance** confirmed that `min_samples_leaf` was the dominant control for model complexity, followed by `max_depth`.
- **Optimization history** showed a healthy exploration phase, followed by stable exploitation and convergence around F1 ≈ 0.78.
- **Parallel coordinate plots** revealed the consistent high-performance region: small leaf sizes (1–2) and moderate depth (12–18).

These insights confirm that CMA-ES navigates the search space efficiently, avoids overfitting, and converges toward interpretable, robust hyperparameter configurations.

Overall Conclusion

Combining both phases, this task demonstrates the power of systematic hyperparameter optimization in machine learning pipelines. The dynamic feature set (from Task 3) and carefully tuned Random Forest yielded large performance gains, while **Bayesian and evolutionary optimization** proved far superior to naive search. Among the tested strategies, **CMA-ES** offered the best trade-off between accuracy and efficiency, and the **custom Gaussian Process optimizer** validated the theory and practice of Bayesian optimization in a transparent way. In summary, structured, data-efficient search combined with informative features enables robust model generalization and reliable performance improvements in real-world sensor-based classification tasks.