Hyperparameter Tunability Analysis of Machine Learning Algorithms

(Automated Machine Learning - Project 1)

Ada Wojterska, Katarzyna Skoczylas, Miłosz Zieliński

18.11.2025

1 Introduction

The goal of this project was to analyze the hyperparameter tunability of the selected machine learning algorithms using at least four datasets. We compared different sampling-based tuning methods and examined how many iterations are needed to achieve stable optimization results. Additionally, we evaluated the overall tunability of each algorithm.

2 Methodology Description

2.1 Datasets

We used four datasets related to medical topics - heart attacks, diabetes, breast cancer and alzheimer from Kaggle. Each dataset contained between 800 and 6900 samples. The data was divided into training and test sets using a 3:1 ratio. In a Jupyter notebook called `Data_Processing`, we performed data preprocessing. Since extensive preprocessing was not a crucial part of our experiment, we applied one-hot encoding for categorical features and Min-Max scaling for numerical features. At first, our ROC-AUC scores were unexpectedly high (around 0.98). After checking the correlations, we decided to modify our datasets by removing the most influential columns.

2.2 Selected Machine Learning Algorithms and Hyperparameter Range Selection

We selected three classification algorithms: XGBoost, K-Nearest Neighbors (KNN), and Decision Tree. These models were chosen because they represent different types of learning methods and are commonly used in practice. Together, they provide a diverse set of approaches for comparing hyperparameter tuning behavior.

The ranges of hyperparameters for all algorithms were chosen according to the values described in article [1]. For the KNN algorithm, additional hyperparameters were included to extend the search space: `weights = ['uniform', 'distance']` defining the neighbor weighting method, and `p = randint(1, 3)` allowing the use of both Manhattan and Euclidean distance metrics. In the Decision Tree algorithm, the hyperparameter cp (complexity parameter) was reduced from (0, 1) to (0, 0.1) to allow the model to grow more detailed trees. In XGBoost, the ranges of

some hyperparameters were narrowed to shorten the computation time and exclude irrational cases where the trees would develop in an unreasonable way.

2.3 Tuning Methods

We selected two hyperparameter tuning methods: `RandomizedSearchCV` from `sklearn.model_selection` for random search and `BayesSearchCV` from `scikit-optimize` for Bayesian optimization. In the randomized search, we set a random seed to ensure that the same hyperparameter grid was used for each dataset and algorithm.

2.4 Experimental Procedure

The Preprocessing, as mentioned before, was performed in a single file. We used ROC-AUC as the evaluation metric for all algorithms. For each of the tuning methods and each dataset, we performed 100 iterations and selected the best set of hyperparameters based on the performance of the test set. In the randomized search, we also calculated the *Star* set of hyperparameters, which represented the mean of the best hyperparameters across all datasets. We then computed the ROC-AUC scores for all datasets using the *Star* hyperparameters and calculated the differences in ROC-AUC between each configuration and the *Star* configuration. For Bayesian optimization, this was not applicable, as the idea behind Bayesian optimization is to iteratively update the hyperparameter distribution based on prior observations rather than using fixed averaged values. We also calculated the test scores for the default models in order to analyze whether our optimized defaults performed better than the package-provided defaults.

3 Tuning Results and analysis of Algorithm Tunability

The table below summarize the ROC-AUC scores obtained for the three analyzed models. It compares the best results from Random Search, Bayesian optimization, *Star* configuration and the default settings provided by the original packages.

Table 1: Comparison of ROC AUC results across all models and datasets

Model	Dataset	Default ROC AUC	Star ROC AUC	Random ROC AUC	Bayes ROC AUC
Decision Tree	alzheimer	0.7159	0.8670	0.8624	0.8679
	cancer	0.7025	0.7790	0.8063	0.8317
	diabetes	0.7041	0.7846	0.7952	0.8332
	heart	0.6500	0.7689	0.7858	0.7858
KNN	alzheimer	0.6769	0.7376	0.7535	0.7535
	cancer	0.7352	0.7726	0.7804	0.7804
	diabetes	0.7128	0.8006	0.8125	0.8125
	heart	0.6699	0.6741	0.6866	0.6846
XGBoost	alzheimer	0.8481	0.8553	0.8648	0.8693
	cancer	0.8261	0.8625	0.8670	0.8675
	diabetes	0.7940	0.8069	0.8324	0.8302
	heart	0.7682	0.7933	0.8020	0.8033

The presented tables show that all models clearly benefited from hyperparameter tuning. For each algorithm, the default configuration achieved the lowest ROC-AUC score, while the Bayesian and Random Search methods produced the best results, with Bayesian optimization slightly outperforming the others in most cases. This suggests that automated tuning improves model performance compared to default settings.

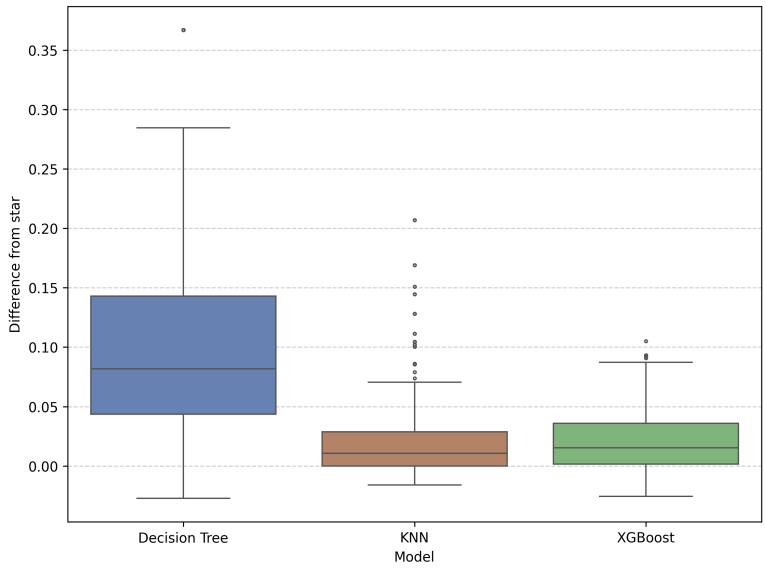

Figure 1: Difference between *Star* and Random Search scores across models.

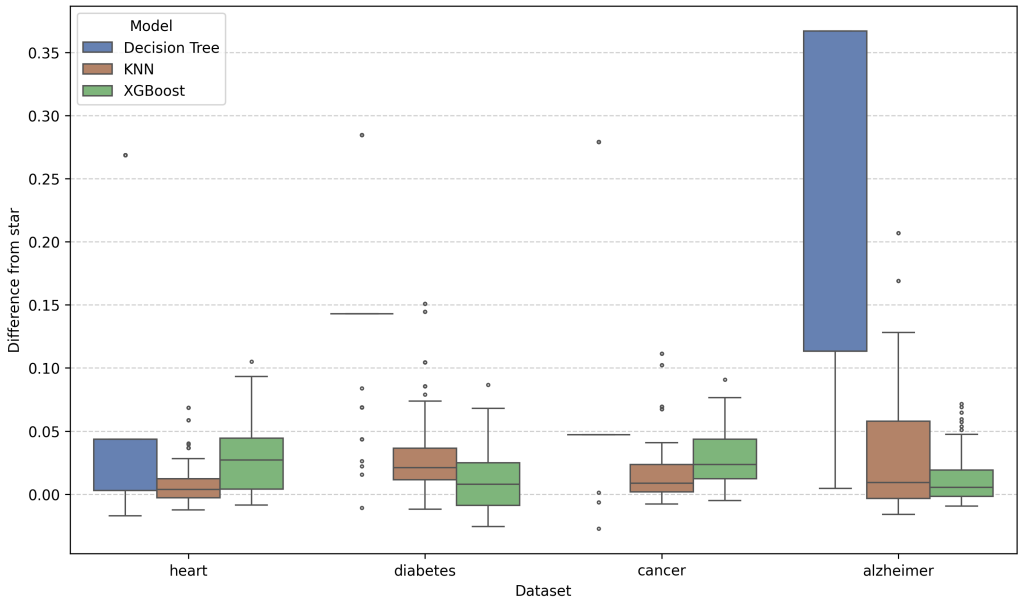

Figure 2: Difference between *Star* and Random Search scores across models and dataset.

The plots above reveal substantial variation in ROC AUC between randomly sampled hyperparameter configurations and the computed *Star* configuration. This indicates that model performance is sensitive to the choice of hyperparameters. Combined with the differences observed across datasets and algorithms, the results suggest that both the data characteristics and the hyperparameter space have a strong influence on the final model performance. Decision Trees exhibit the largest variability, while XGBoost and KNN remain relatively stable. This observation is reinforced in Figures 4 and 5, which illustrate how performance evolves with increasing numbers of iterations for both sampling strategies. XGBoost is visibly more stable than the other two algorithms, and the random search method tends to be slightly less consistent than Bayesian optimization. However, the plots also show that Bayesian optimization does not guarantee improvement at every iteration. For this reason, we analyzed the specific iterations in which Bayesian optimization provides

performance gains. As shown in Figure 3, the most substantial improvements occur within the first 10 iterations, and after around 20 iterations the gains become infrequent and marginal.

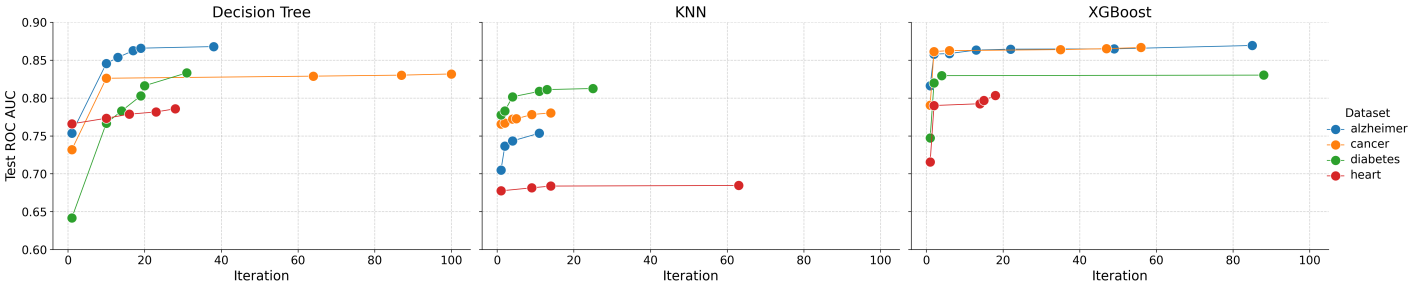

Figure 3: Score improvements across Bayesian optimization iterations

4 Additional research

To extend our experiment, we divided each dataset into 25%, 50%, and 75% subsets and repeated the entire optimization process. As shown in Figure 6, models generally achieved higher scores on larger subsets, although the differences were not substantial. We also investigated whether dataset size affects the number of iterations required for Bayesian optimization to converge. We measured the number of times the model's performance improved and identified the iteration at which the last improvement occurred (considering only improvements greater than 0.01). To summarize these findings, we computed the mean and standard deviation of the iteration at which the model's last significant performance improvement occurred. The results in Table 2 indicate a tendency for smaller datasets (particularly at 25% size) to require more optimization steps, although the trend varied across datasets and algorithms.

Table 2: Mean and standard deviation of last improvements in Bayesian optimalization for different models.

Size	Decision Tree		KNN		XGBoost	
	Mean	Std	Mean	Std	Mean	Std
25	28.50	36.318	17.00	27.346	29.00	45.056
50	11.50	3.000	2.25	1.258	2.25	0.500
75	10.00	0.000	10.25	9.946	2.00	0.000
100	13.00	12.728	4.25	4.717	2.00	0.000

Literature

[1] Smith, J., & Doe, A. (2019). *The Tunability of Machine Learning Models*. Journal of Machine Learning Research, 20(18), 1–15. Access online: `https://jmlr.org/papers/volume20/18-444/18-444.pdf`

List of Appendices

1. Appendix A – Additional plots and visualizations of model performance
2. Appendix B – Example hyperparameter configurations for each dataset
3. Appendix C – Source code and raw CSV experiment results are available in the project repository at: GitHub Repository

1. Appendix A – Additional plots and visualizations of model performance

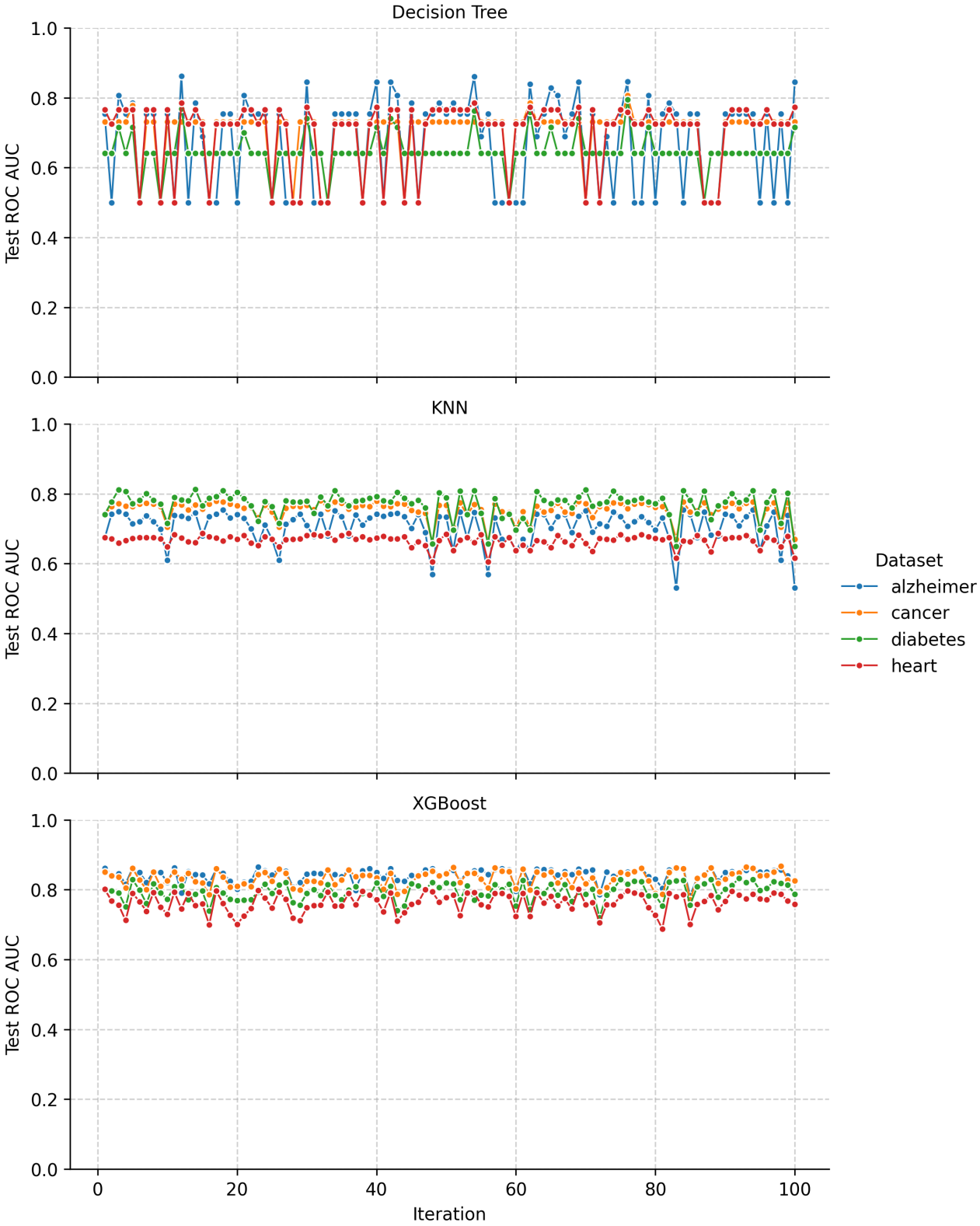

Figure 4: Comparison of ROC AUC across models (Random Search)

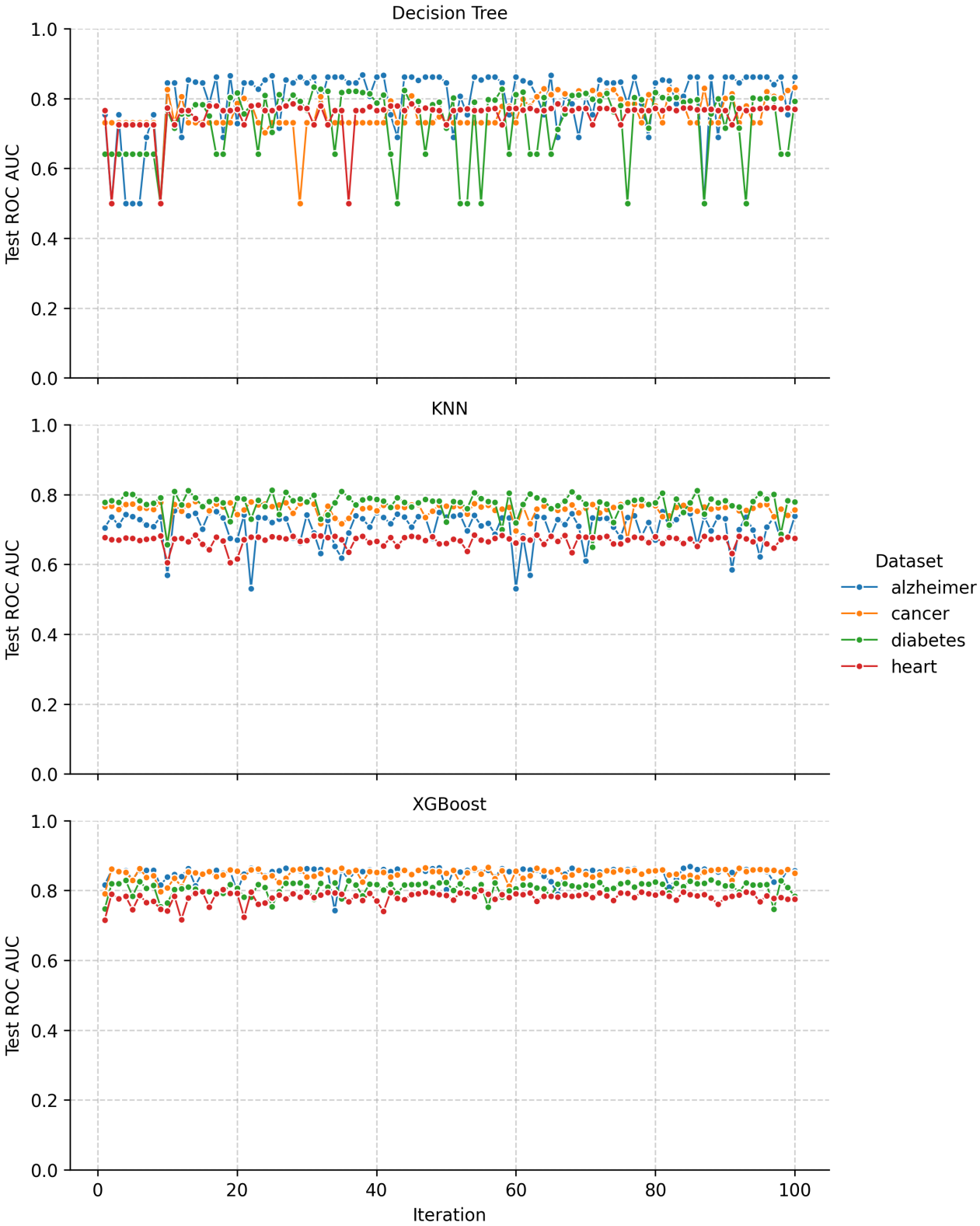

Figure 5: Comparison of ROC AUC across models (Bayesian Method)

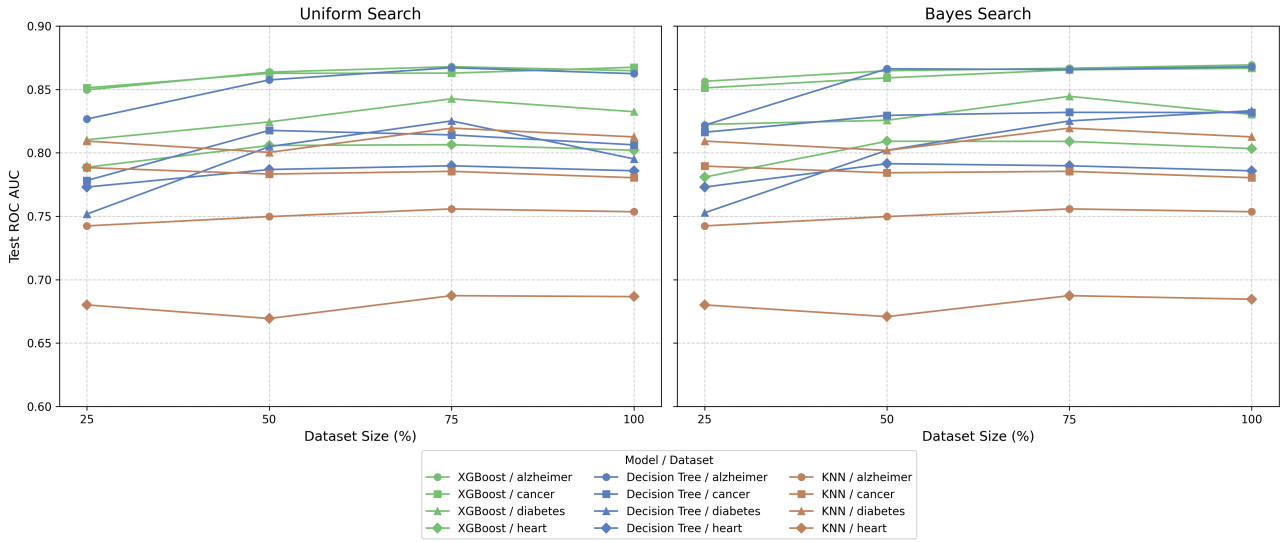

Figure 6: Comparison of ROC AUC across datasets and models - subsets

2. Appendix B – Example hyperparameter configurations for each dataset

Table 3: Decision Tree – best hyperparameters per optimization method

Method	Dataset	ccp_alpha	max_depth	min_samples_leaf	min_samples_split
Bayes	alzheimer	0.0014	25	23	60
	cancer	0.0002	29	52	57
	diabetes	0.0000	19	6	42
	heart	0.0056	30	60	2
Uniform	alzheimer	0.0065	4	25	15
	cancer	0.0005	14	18	3
	diabetes	0.0005	14	18	3
	heart	0.0065	4	25	15
	STAR	0.0035	9	22	9

Table 4: KNN – best hyperparameters per optimization method

Method	Dataset	n_neighbors	p	weights
Bayes	alzheimer	30	1	distance
	cancer	30	2	distance
	diabetes	27	1	uniform
	heart	19	1	distance
Uniform	alzheimer	30	1	distance
	cancer	30	2	distance
	diabetes	27	1	uniform
	heart	10	2	distance
	STAR	24	1	distance

Table 5: XGBoost – best hyperparameters per optimization method

Method	Dataset	colsample_bylevel	colsample_bytree	learning_rate	max_depth	min_child_weight	n_estimators	reg_alpha	reg_lambda	subsample
Bayes	alzheimer	0.562	1.000	0.006	5	5.266	100	0.031	0.031	0.628
	cancer	0.919	1.000	1.000	4	32.000	2000	32.000	8.976	0.799
	diabetes	0.836	0.877	0.096	3	1.000	862	9.028	32.000	0.503
	heart	0.534	0.591	0.212	9	18.607	1865	31.507	32.000	0.500
Uniform	alzheimer	0.833	0.667	0.012	10	22.359	500	22.427	1.863	0.722
	cancer	0.556	0.889	0.322	3	3.723	694	18.775	0.076	0.667
	diabetes	0.944	1.000	0.712	11	12.301	338	5.411	0.537	1.000
	heart	1.000	0.667	0.083	10	28.395	994	26.789	0.031	0.611
	STAR	0.833	0.806	0.282	8	17.000	632	18.351	0.627	0.750