AutoML project: Tuning a majority voting ensemble consisting of random forest classifier, gradient boosting classifier and SVM classifier using DEHB

Constantin von Crailsheim[1]

[1]LMU Munich, Institute of Statistics

Abstract The AutoML system tunes a ML pipeline of an imputer, an optional sampler, an optional standardizer and a choice of three models. Since the tasks are imbalanced tabular data, a random forest classifier, a gradient boosting classifier and a SVM classifier are reasonable choices, which should yield a good overall prediction by majority voting. I used DEHB as an optimizer, which combines the advantages of differential evolutions and hyperband, i.e., generating promising new hyperparameter configurations for longer training based on initial cheap evaluations. Overall, the improvement in performance compared to the untuned random forest classifier baseline is 0.6% to 25.3%. Thus the AutoML system outperforms the baseline across all datasets. The source code is available at: `https://github.com/constantin-crailsheim/automl_imbalanced`

1 Introduction

The objective of the AutoML system is to yield good performance as measured by the balanced accuracy on imbalanced tabular datasets and be able to deal with missing values. Thus, it optimizes a ML pipeline consisting of five elements. First, a choice of two imputers replaces missing values. Then, an optional sampling method is applied to deal with imbalance in the targets, which consists of both under- and oversampling methods. Next, the pre-processed integer features are rounded such that they do not take values that would not appear in the training data. After optionally normalizing all features, three different models were fitted, i.e., a random forest classifier, a gradient boosting classifier and a SVM classifier. The choice of pre-processing and the hyperparameters of the model were optimized by DEHB, which is a computationally cheap optimizer that works well on discrete search spaces. The final predictions are derived by majority voting of the individual predictions of the best pipelines for each model. This means that the overall predictions should be good of the errors of the model are not too correlated. I used a M1 Pro chip to run the optimization.

2 Method

This section outlines the specific choices of the AutoML system and how it will be optimized and evaluated. The choices of the imputation strategy of sklearn.impute are:

- The SimpleImputer is an univariate imputer, which completes missing values with a descriptive statistic per feature. I chose the median, since it is less sensitive to outliers than the mean.
- The KNNImputer replaces missing values by the mean value of its 5 nearest neighbors as default based on Euclidian distance of their non-missing observations of the same feature.

The choices of data-level sampling method to yield balanced dataset of imblearn are:

- SMOTE as an oversampling approach generates new samples of the minority class by interpolating between existing observations of the minority class, where no distinction is made between easy and hard samples.

- TomekLinks as an undersampling approach removes samples from the majority class, if they are nearest neighbors to a minority class sample, thus removing noisy borderline samples of the majority class.

- SMOTETomek, which combines SMOTE and Tomek links.

- No sampling method, which would allow algorithmic-level methods to deal with the imbalanced data.

The above pre-processing methods could generate numeric values for features that should contain only integers (e.g., ordinal categorical features), since they impute missing values by means or medians and generate new samples by interpolation. Thus, another layer is added to the pipeline, which rounds all observations of those features to an integer.

The last step of pre-processing is the choice of whether to apply the StandardScaler of sklearn.preprocessing to standardize the features. This will be in particular useful for the SVM, since an RBF kernel assumes features centered around zero and similar variance across features.

Subsequently, the hyperparameters of the three models of the ensemble were optimized, where the search space was defined with the ConfigSpace package. For almost all hyperparameters, the default was set to the default specified for each model. The search space was mostly chosen such that it is centered around the default while accounting for log scale. In those cases where the default would be a the lower end of a reasonable search space, the upper bound was chosen higher.

The first model in the ensemble is a RandomForestClassifier from sklearn.ensemble. It can handle all data types well and generalizes well by having a low variance due to ensembling over relatively uncorrelated models. The hyperparameters, which will all be sampled uniformly, are:

Hyperparameter	Data type	Search space	Default	Other
criterion	Categorical	{Gini, Entropy, Log loss}	Gini	
max_depth	Integer	[5,15]	10	
min_samples_split	Integer	[1, 32]	2	Log scale
min_samples_leaf	Integer	[1, 16]	1	Log scale
max_features	Integer	[0.1, 0.9]	0.5	
class_weight	Categorical	{Balanced, Balanced subsample, None}	None	

The class_weight is an algorithm-level method that deals with imbalanced data and will only have an effect if no data-level sampling method was used, since in that case the dataset passed to the model will be balanced.

The second model in the stack is a GradientBoostingClassifier from sklearn.ensemble. It has strong predictive performance by iteratively fitting weak learners on the error of the previous learner and has similar advantages as the RandomForestClassifier. The hyperparameters are:

Hyperparameter	Data type	Search space	Default	Other
loss	Categorical	{Log loss, Exponential}	Log loss	
learning_rate	Float	[0.01, 1]	0.1	Log scale
criterion	Categorical	{Friedman MSE, Squared error}	Friedman MSE	
min_samples_split	Integer	[1, 32]	2	Log scale
min_samples_leaf	Integer	[1, 16]	1	Log scale
max_depth	Integer	[2,10]	3	

The third model in the stack is a Support Vector Classifier (SVC) from sklearn.svm. This model works in particular well on easily to separate datasets and in high-dimensional spaces [1]. The hyperparameters are:

Hyperparameter	Data type	Search space	Default	Other
C	Float	[0.1, 10]	1.0	Log scale
kernel	Categorical	{Linear, Polynomial, RBF, Sigmoid}	RBF	
shrinking	Boolean	{True, False}	True	
tol	Float	[1e-4,1e-2]	1e-3	
class_weight	Categorical	{Balanced, None}	None	

To optimize the hyperparameter of the AutoML system, I used DEHB by Awad et al. (2021), which combines differential evolution and hyperband. Differential evolution constructs a new mutant vector from three random parents and then generates the offspring by randomly selecting values from the new mutant vector with probability p and otherwise from one of the corresponding parents. Hyperband allows to search the whole search space with cheap evaluations and only trains more costly models on promising areas of the search space. The algorithm starts by sampling N random hyperparameter configurations, which are evaluated at the lowest budget. Then the best $1/\eta$ of these configurations are evaluated at a η-times higher budget and this process is repeated until the highest fidelity (denoted here by f) is reached, thus $N = \eta^{f-1}$. After completing one iteration, the algorithm restarts with new instantiations and evaluates these at the second lowest fidelity, thereby hedging against bad initializations. DEHB combines both approaches by generating the hyperparameter configurations for the next fidelity by differential evolution from the lower fidelity as parent pool. The authors state that DEHB is computationally cheap with high speed-up gains compared to BOHB. Furthermore, it has strong final performance for discrete search spaces, which I have for various hyperparameters. The authors experiments have shown that DEHB also outperforms SMAC by mean ranks across all chosen benchmarks. For those reasons, I chose DEHB as an efficient optimizer for this problem.

For the optimization, I set $\eta = 3$ and $f = 4$, which implies an initial population of $N = 3^3 = 27$. I set the budget for the RandomForestClassifier and the GradientBoostingClassifier, as indicated by the number of trees in the forest, to a minimum of 10 and maximum of 270. For SVC, the budget is indicated by the maximum number of iterations and I set it to a minimum of 500 and maximum of 13500. However, the runtime between the lowest and largest budget did not differ too much since the SVM optimizer most likely already converged in most cases and the evaluation was relatively cheap compared to the forest based classifiers for most datasets. Thus, the SVC mostly benefits from differential evolution and the successive halving element is not as important since many configurations can be tested irrespecitively. Hence, I allocated 40% of the maximum cost to optimizing the forest based models and 20% to optimizing the SVC.

To evaluate the performance of the AutoML system, I used 3-fold external and 4-fold internal cross-validation. Given a total budget of 3600 seconds per dataset, a total of 1200 second could be used to optimize the AutoML system in each fold. Since the budget is not that larger after accounting for cross-validation, I tuned only a selection of hyperparameters of the actual model and the hyperparameters of the preprocessing functions were kept at their default values.

[1] https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107, Accessed on 26/03/2023

After the optimization routine, each model is fitted with incumbent configuration and the unbalanced sampling is removed from the pipeline to not sample from the test set. The final AutoML system is an ensemble of three models with majority voting for the final classification.

3 Experiments

The external cross-validation performance in terms of balanced accuracy of the AutoML system vs. the untuned random forest baseline for each dataset id is shown below:

Model	976	980	1002	1018	1019	1021	1040	1053	1461	41160
Baseline	0.965	0.936	0.543	0.546	0.990	0.918	0.963	0.592	0.695	0.571
AutoML system	0.989	0.982	0.788	0.799	0.996	0.957	0.992	0.661	0.784	0.667
Improvement	0.023	0.046	0.245	0.253	0.006	0.038	0.029	0.069	0.089	0.095

Hence, the AutoML system outperforms the baseline across all datasets with an improvement between 0.6% and 25.3%. Include test?

The plots of the trajectories for each dataset are in appendix A. For dataset 976, 980 1019, the SVC is the top performing estimator, followed by the GradientBoostingClassifier and the RandomForestClassifier. For the other datasets, the ranking is not as clear. For the last two datasets, the SVC has the worst performance, most likely since SVCs are costlier to fit given the large number of observations and less iterations to tune the hyperparameters were possible. This probably also compromise the overall performance of the AutoML system. The benchmarks are usually outperformed after at most 80 seconds, except for the last two datasets due to the underperforming SVC. The final externally cross-validated performance tends to be a bit lower than the performances of the individual algorithms. This probably arises, since the performance of the individual algorithms is overly optimistic due to the hyperparameters being tuned on that specific fold.

Write sth about final performance.

The chosen incumbents for each model for all datasets are shown in the appendix. The key trends are as following: The preferred sampling method for RandomForestClassifier and SVC is quite mixed, but for the GradientBoostingClassifier only oversampling or mixed methods were considered better. As expected, SVC uses the StandardScaler for all datasets except 1019 (Why?). If no sampling method was chosen, the imbalanced was always accounted for by class weights.

4 Conclusion

The AutoML system outperforms the benchmark across all datasets and has thus proven to be useful. However, some improvements are still possible if a larger budget would be available. The hyperparameters of the pre-processing methods could be also tuned to fit better to each dataset and algorithm. Furthermore, a stacking classifier could be trained on the three algorithms, which would allow to find the best combination of the predictions of each individual algorithm for the final prediction of the AutoML system. This would be in particular beneficial to the last two datasets, where the ensemble performance was significantly worse than the two best individual performances. I conducted experiments with stacking, but the training on the final incumbents with the highest budgets took fairly long, thus I decided to use the budget rather to improve the performance of the individual estimators.

References

Awad, N., Mallik, N., and Hutter, F. (2021). DEHB: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In Zhou, Z., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2147–2153. ijcai.org.

A Trajectories

The following plots show the trajectories of the incumbent performance for all three models evaluated with internal cross-validation in each fold over runtime. Furthermore, the external cross-validation of the untuned random forest classifier baseline is plotted as grey horizontal line and the external cross-validation of final performance of the AutoML system is plotted as purple dot at the maximum runtime per model in each fold.

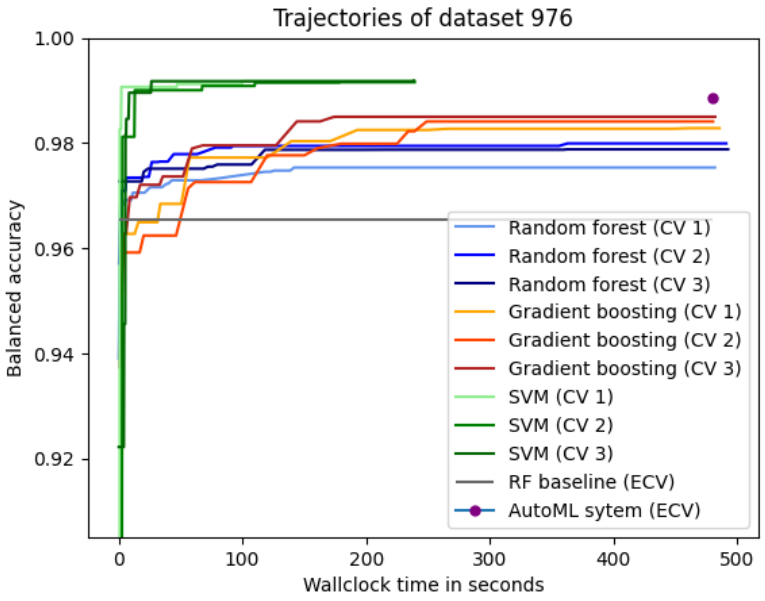

Figure 1: Trajectories for dataset 976 with 9961 observations and 14 features.

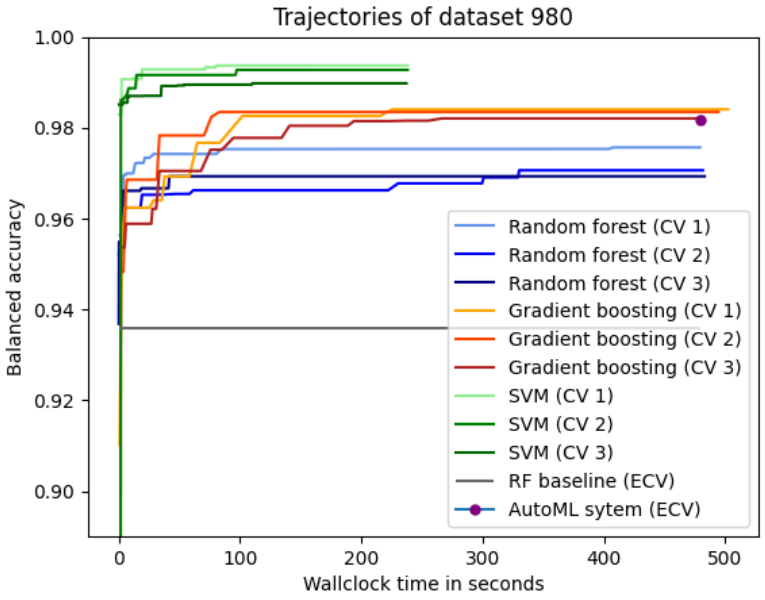

Figure 2: Trajectories for dataset 980 with 5620 observations and 64 features.

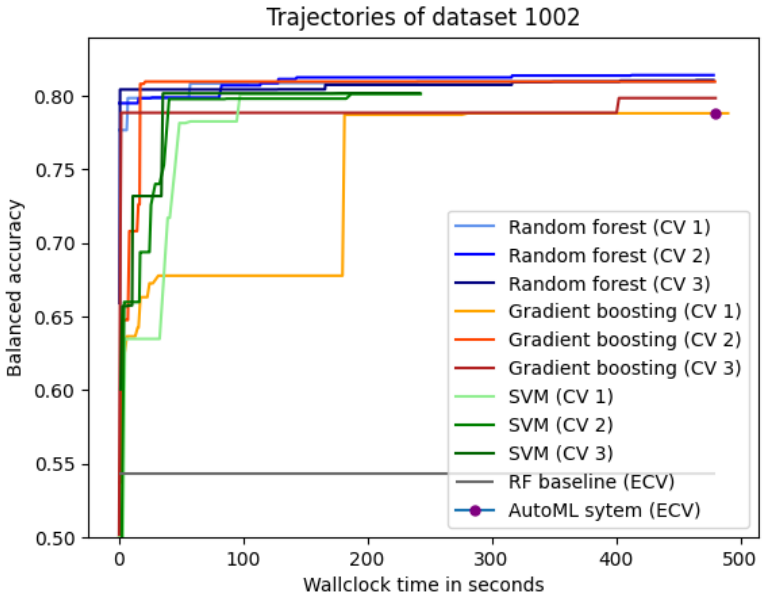

Figure 3: Trajectories for dataset 1002 with 7485 observations and 55 features.

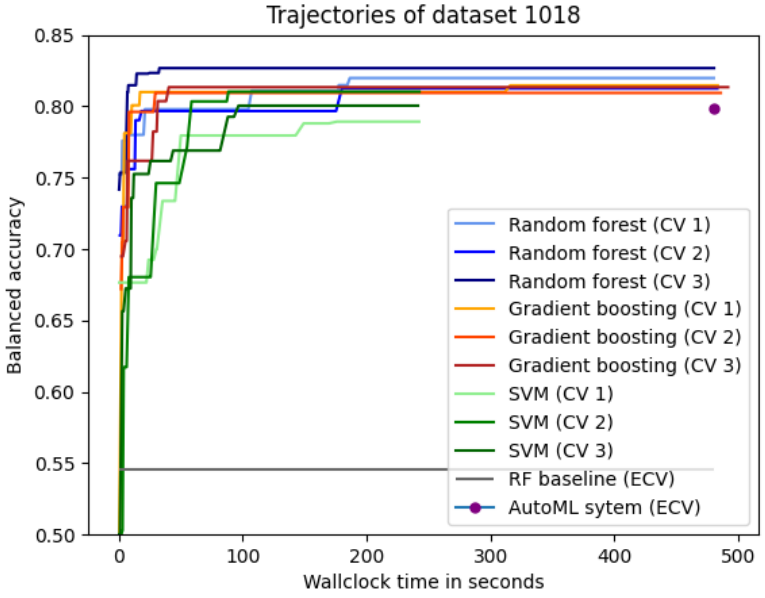

Figure 4: Trajectories for dataset 1018 with 8844 observations and 56 features.

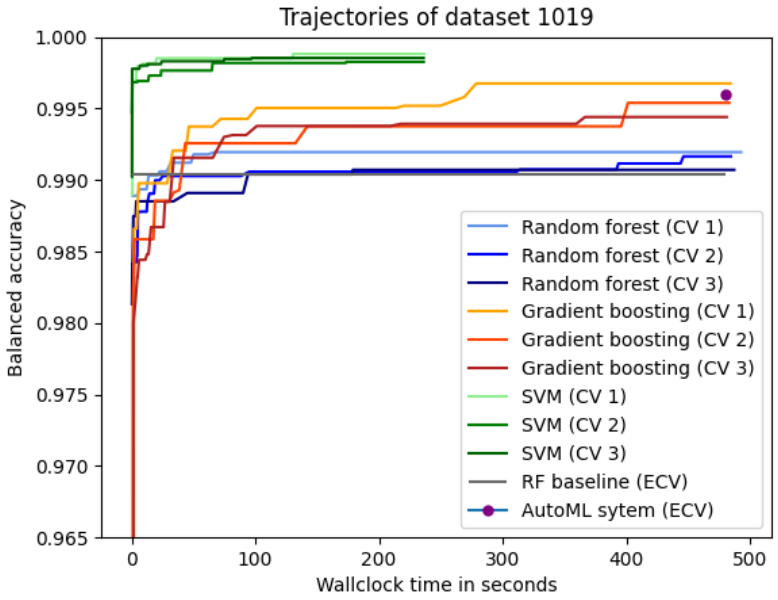

Figure 5: Trajectories for dataset 1019 with 10992 observations with 16 features

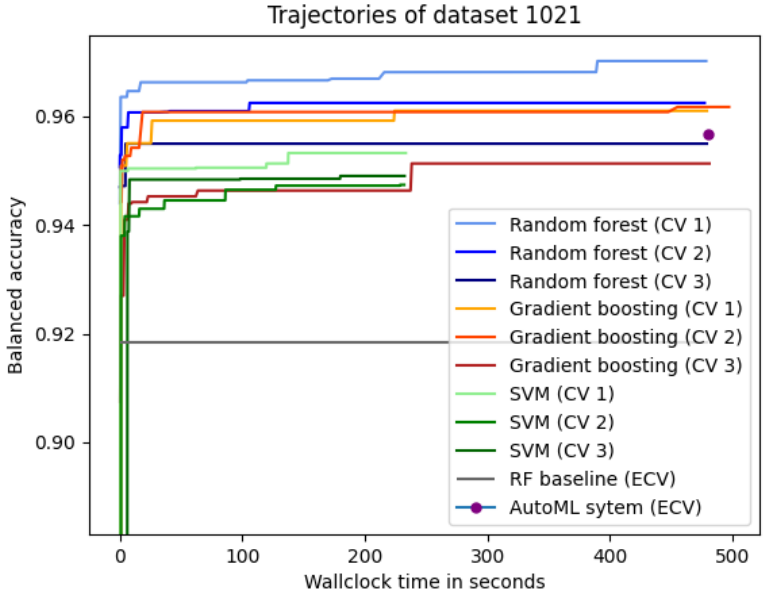

Figure 6: Trajectories for dataset 1021 with 5473 observations and 10 features.

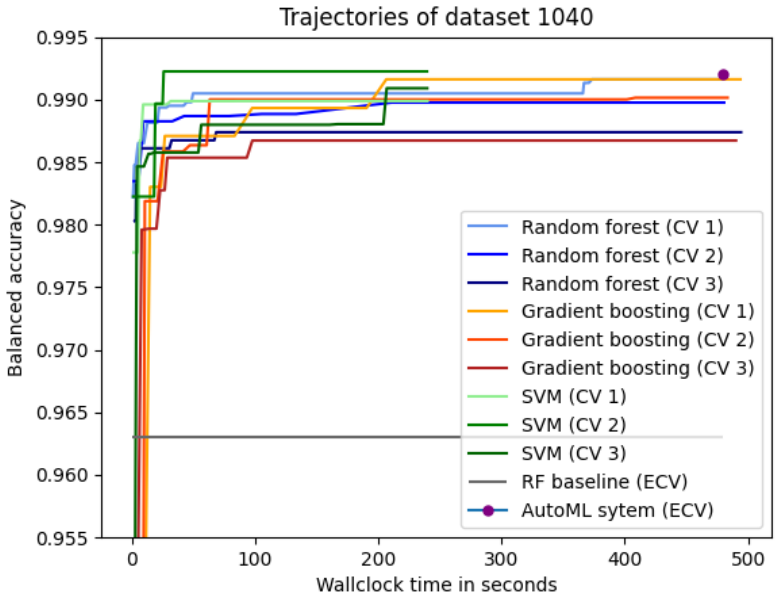

Figure 7: Trajectories for dataset 1040 with 14395 observations and 108 features.

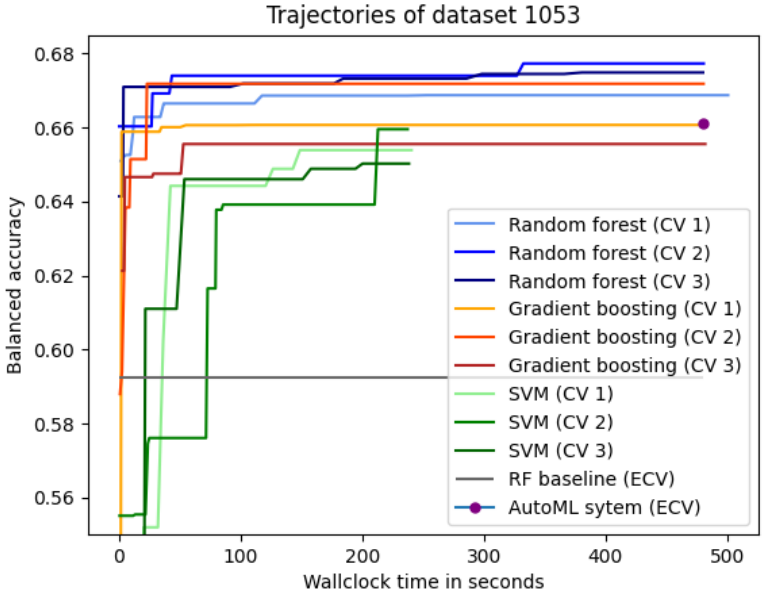

Figure 8: Trajectories for dataset 1053 with 10885 observations and 21 features.

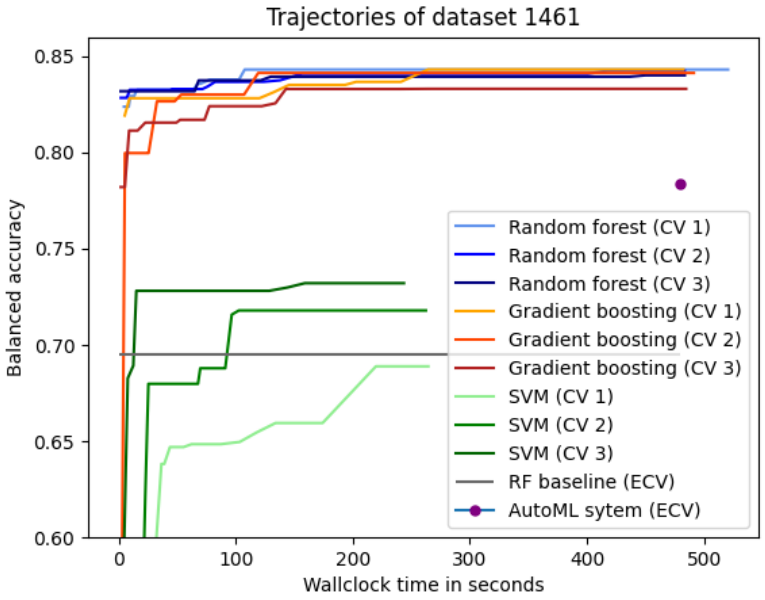

Figure 9: Trajectories for dataset 1461 with 45221 observations and 16 features.

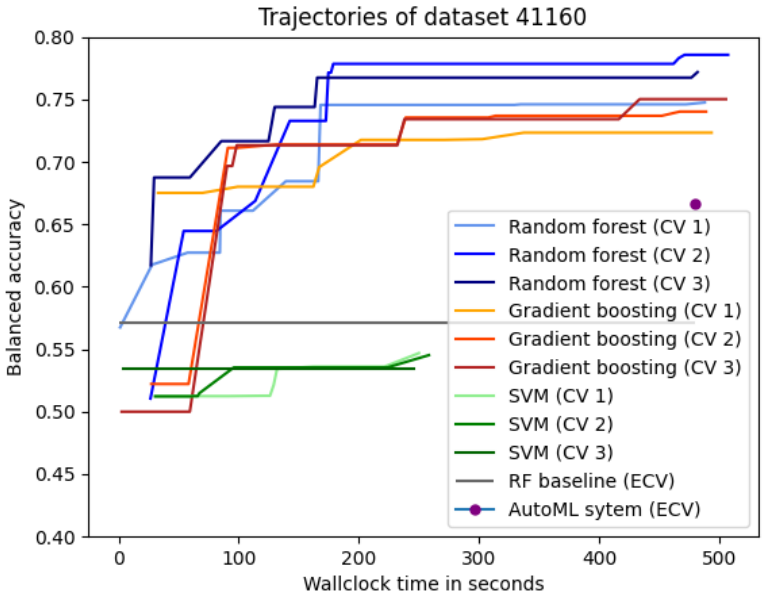

Figure 10: Trajectories for dataset 41660 with 31406 observations and 22 features.

B Incumbent hyperparameter

The following tables shows the incumbent hyperparameter configurations for each algorithm and each dataset in each fold.

Table 1: Incumbent hyperparameter configurations for the random forest classifier

Dataset ID	CV fold	Imputer	Sampler	Scaler	Criterion	Max depth	Min samples per split	Min samples per leaf	Max features	Class weight
976	1	Simple	SMOTETomek	False	entropy	22	4	1	0.656466	None
976	2	Simple	SMOTETomek	True	log_loss	17	4	1	0.115289	balanced_subsample
976	3	Simple	SMOTE	False	entropy	16	4	2	0.262840	balanced
980	1	Simple	SMOTETomek	False	entropy	16	2	2	0.457415	None
980	2	KNN	SMOTETomek	True	entropy	19	2	9	0.282039	balanced
980	3	Simple	SMOTETomek	True	log_loss	11	1	6	0.144065	None
1002	1	KNN	Tomek	False	entropy	5	8	2	0.228523	balanced_subsample
1002	2	Simple	None	False	log_loss	5	5	9	0.392731	balanced_subsample
1002	3	Simple	Tomek	True	gini	6	11	4	0.192183	balanced
1018	1	KNN	Tomek	False	entropy	6	1	14	0.105286	balanced
1018	2	KNN	Tomek	False	log_loss	5	25	3	0.497648	balanced_subsample
1018	3	KNN	Tomek	True	gini	5	9	6	0.285309	balanced
1019	1	Simple	SMOTETomek	True	entropy	23	2	1	0.806836	balanced_subsample
1019	2	KNN	SMOTETomek	False	entropy	19	3	1	0.195487	None
1019	3	Simple	SMOTE	True	log_loss	24	1	5	0.726165	None
1021	1	KNN	SMOTE	False	gini	20	19	4	0.134323	balanced_subsample
1021	2	Simple	SMOTE	False	entropy	22	9	9	0.495168	balanced
1021	3	Simple	SMOTETomek	True	entropy	24	14	3	0.208502	None
1040	1	Simple	SMOTETomek	False	gini	8	11	4	0.577811	balanced_subsample
1040	2	Simple	SMOTETomek	False	gini	23	2	5	0.797167	None
1040	3	KNN	Tomek	False	gini	5	1	1	0.390420	balanced
1053	1	KNN	SMOTETomek	False	log_loss	9	2	4	0.393722	balanced_subsample
1053	2	KNN	Tomek	True	log_loss	7	9	4	0.791928	balanced
1053	3	Simple	Tomek	True	entropy	18	4	12	0.656912	balanced_subsample
1461	1	KNN	Tomek	False	log_loss	20	1	11	0.158737	balanced_subsample
1461	2	KNN	SMOTETomek	True	entropy	11	21	4	0.517657	balanced
1461	3	KNN	Tomek	False	entropy	14	31	2	0.221436	balanced
41160	1	Simple	None	False	entropy	18	4	6	0.769477	balanced
41160	2	Simple	Tomek	True	entropy	10	28	3	0.783361	balanced
41160	3	Simple	Tomek	True	gini	12	3	4	0.808427	balanced

Table 2: Incumbent hyperparameter configurations for the gradient boosting classifier

Dataset ID	CV fold	Imputer	Sampler	Scaler	Loss	Learning rate	Criterion	Min samples per split	Min samples per leaf	Max depth
976	1	Simple	SMOTE	True	exponential	0.814630	friedman_mse	3	2	8
976	2	Simple	SMOTETomek	True	exponential	0.979669	friedman_mse	6	8	10
976	3	KNN	SMOTETomek	False	exponential	0.948759	friedman_mse	4	11	8
980	1	Simple	SMOTE	True	log_loss	0.360042	friedman_mse	4	9	4
980	2	KNN	SMOTETomek	True	exponential	0.359648	squared_error	10	1	4
980	3	KNN	SMOTETomek	True	exponential	0.366579	squared_error	24	12	4
1002	1	KNN	SMOTETomek	True	exponential	0.023344	friedman_mse	5	1	3
1002	2	KNN	SMOTE	True	log_loss	0.016664	friedman_mse	2	5	2
1002	3	Simple	SMOTETomek	True	log_loss	0.026060	friedman_mse	4	3	3
1018	1	Simple	SMOTE	True	exponential	0.039226	friedman_mse	9	13	2
1018	2	Simple	SMOTETomek	False	exponential	0.027759	squared_error	21	1	4
1018	3	KNN	SMOTE	False	log_loss	0.065935	friedman_mse	13	6	3
1019	1	KNN	SMOTETomek	True	exponential	0.566058	friedman_mse	11	5	4
1019	2	KNN	SMOTE	True	exponential	0.686387	squared_error	10	1	6
1019	3	Simple	SMOTETomek	True	log_loss	0.358116	squared_error	7	2	4
1021	1	Simple	SMOTE	True	log_loss	0.230746	friedman_mse	24	8	5
1021	2	Simple	SMOTE	False	exponential	0.020508	friedman_mse	12	6	11
1021	3	KNN	SMOTETomek	False	log_loss	0.283118	squared_error	5	2	3
1040	1	Simple	SMOTE	True	exponential	0.428100	friedman_mse	14	1	3
1040	2	KNN	SMOTE	True	exponential	0.690849	friedman_mse	11	4	4
1040	3	KNN	SMOTETomek	True	exponential	0.545570	friedman_mse	5	5	2
1053	1	Simple	SMOTETomek	True	exponential	0.031338	friedman_mse	3	10	5
1053	2	Simple	SMOTE	True	log_loss	0.010315	friedman_mse	8	8	2
1053	3	Simple	SMOTETomek	False	exponential	0.038175	friedman_mse	5	6	2
1461	1	KNN	SMOTETomek	False	exponential	0.097863	friedman_mse	3	4	5
1461	2	KNN	SMOTETomek	True	exponential	0.129724	friedman_mse	23	4	7
1461	3	KNN	SMOTE	True	exponential	0.092295	squared_error	26	14	6
41160	1	KNN	SMOTE	False	log_loss	0.018754	friedman_mse	4	11	14
41160	2	Simple	SMOTETomek	False	log_loss	0.025380	friedman_mse	5	2	8
41160	3	Simple	SMOTE	False	exponential	0.011618	squared_error	5	16	10

Table 3: Incumbent hyperparameter configurations for the SVM classifier

Dataset ID	CV fold	Imputer	Sampler	Scaler	C	Kernel	Shrinking	Tolerance	Class weight
976	1	KNN	None	True	6.812195	rbf	True	0.000136	balanced
976	2	Simple	SMOTETomek	True	5.942369	rbf	True	0.002453	balanced
976	3	Simple	Tomek	True	5.042524	rbf	False	0.000554	balanced
980	1	Simple	SMOTETomek	True	2.461050	poly	False	0.004004	balanced
980	2	Simple	SMOTE	True	0.810882	poly	True	0.002291	balanced
980	3	KNN	Tomek	False	1.221203	rbf	False	0.003969	balanced
1002	1	KNN	Tomek	True	0.230800	linear	False	0.002982	balanced
1002	2	KNN	SMOTETomek	True	0.127835	sigmoid	True	0.001461	balanced
1002	3	Simple	SMOTETomek	True	0.384752	linear	False	0.005138	balanced
1018	1	Simple	SMOTE	True	0.202185	sigmoid	False	0.000893	None
1018	2	Simple	None	True	0.245091	linear	True	0.001234	balanced
1018	3	Simple	SMOTETomek	True	0.135464	linear	False	0.000205	balanced
1019	1	KNN	Tomek	False	9.389382	poly	False	0.000422	None
1019	2	KNN	SMOTETomek	False	1.538617	poly	False	0.000794	None
1019	3	KNN	SMOTETomek	False	9.673782	rbf	True	0.002117	balanced
1021	1	KNN	None	True	7.955140	rbf	False	0.000758	balanced
1021	2	KNN	None	True	8.527953	rbf	False	0.000206	balanced
1021	3	KNN	Tomek	True	7.187957	rbf	True	0.000581	balanced
1040	1	Simple	SMOTETomek	True	0.506277	linear	True	0.001319	None
1040	2	KNN	Tomek	True	0.126113	linear	False	0.003028	balanced
1040	3	Simple	SMOTETomek	True	1.165825	sigmoid	True	0.002011	None
1053	1	Simple	SMOTETomek	True	2.294233	rbf	False	0.000396	None
1053	2	Simple	Tomek	True	3.810805	rbf	False	0.009082	balanced
1053	3	KNN	SMOTETomek	True	1.594506	linear	False	0.001114	balanced
1461	1	Simple	SMOTETomek	True	0.485222	sigmoid	False	0.000393	None
1461	2	Simple	Tomek	True	3.769543	sigmoid	True	0.001322	balanced
1461	3	Simple	None	True	0.533370	sigmoid	False	0.008771	balanced
41160	1	KNN	Tomek	True	8.007605	rbf	False	0.000958	None
41160	2	KNN	SMOTE	True	0.227120	rbf	True	0.001452	balanced
41160	3	Simple	SMOTE	True	9.758566	sigmoid	False	0.000217	balanced

C Information about datasets

Dataset ID	Observations	Features
976	9961	14
980	5620	64
1002	7485	55
1018	8844	56
1019	10992	16
1021	5473	10
1040	14395	108
1053	10885	21
1461	45221	16
41160	31406	22