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

Constantin von Crailsheim[1]

[1]LMU Munich, Institute of Statistics

Abstract The AutoML system tunes an ensemble of three modeling pipelines with an imputer, an optional sampler, an optional scaler as pre-preprocessing. Since the tasks are imbalanced tabular datasets, a random forest classifier, a gradient boosting classifier and a SVM classifier are reasonable model 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 by generating promising new hyperparameter configurations for more costly evaluations based on initial cheap evaluations. The performance improvement in balanced accuracy compared to the untuned random forest classifier baseline is between 0.8% and 25.9%. Thus, the AutoML system outperforms the baseline across all datasets and the algorithm is significantly different for 9 out of 10 datasets using the McNemar test. The source code is available at: `https://github.com/constantin-crailsheim/automl_imbalanced`

1 Introduction

The objective of the AutoML system is to achieve good performance on imbalanced tabular datasets, measured in terms of balanced accuracy, and to be able to handle missing values. Thus, it optimizes three ML pipelines consisting of five elements. First, a choice of two imputers is used to handle missing values. Then, an optional sampling method is applied to deal with imbalance in the targets, which can be under- and/or oversampling methods. Next, the pre-processed integer features are rounded so that they do not take on values that would not appear in the original data. After optionally standardizing all features, for each pipeline a different models was fitted, i.e., a random forest classifier, a gradient boosting classifier and a SVM classifier. The pre-processing choices 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 over the individual predictions of the best pipelines for each model. This means that the overall predictions should be in particular good if the errors of the models are not too correlated.

2 Method

This section outlines the specific choices of the AutoML system and how it will be optimized and evaluated. The choices of `sklearn.impute` for the imputation strategy 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 (by default) 5 nearest neighbors, as determined by the Euclidean distance of their non-missing observations of the same feature.

The choices of `imblearn` for the data-level sampling method to yield a balanced dataset 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` combines SMOTE and Tomek links.

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

The above pre-processing methods might 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, I added another layer to the pipeline, which rounds all observations of these features to the closest 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 particularly 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 tuned, where I defined the search space with the `ConfigSpace` package. For almost all hyperparameters, the default was set to the default specified for each model. In most cases, I chose the search space to be centered around the default, accounting for log scale. In those cases where the default would be at the lower end of a sensible search space, I chose the upper bound to be reasonably 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 are all uniformly sampled, are:

Hyperparameter	Data type	Search space	Default	Other
`criterion`	Categorical	{`gini`, `entropy`, `log_loss`}	`gini`	
`max_depth`	Integer	[5,25]	15	
`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 by giving more weight to less frequent classes. It will only have an effect if no data-level sampling method has been used, since otherwise the dataset passed to the model will be already balanced.

The second model in the ensemble 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	[2, 32]	2	Log scale
`min_samples_leaf`	Integer	[1, 16]	1	Log scale
`max_depth`	Integer	[2,15]	3	

The third model in the ensemble is a Support Vector Classifier (`SVC`) from `sklearn.svm`. This model works particularly well on easily separable datasets, on small data sets, and in high-dimensional spaces (GeeksforGeeks, 2023). The hyperparameters are:

Hyperparameter	Data type	Search space	Default	Other
`C`	Float	[0.1, 10]	1.0	Log scale
`kernel`	Categorical	{`linear`, `poly`, `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 hyperparameters 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 the whole search space to be searched with cheap evaluations and trains more costly models only on promising regions 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 an iteration, the algorithm restarts with new instantiations and evaluates them 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 of their 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 $N = 3^3 = 27$. For the `RandomForestClassifier` and the `GradientBoostingClassifier`, I set the budget, as indicated by the number of trees in the forest, to a minimum of 10 and a maximum of 270. For `SVC`, I chose the maximum number of iterations as budget and I set it to a minimum of 500 and maximum of 13500. However, the runtime between the lowest and highest budget usually 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 irrespectively. 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. I used stratified CV to ensure that the class imbalance of the targets was preserved in each fold. Given a total budget of 3600 seconds per dataset, a total of 1200 seconds could be used to optimize the AutoML system in each fold. Since the budget is not that large after accounting for cross-validation, I only tuned a selection of hyperparameters of the actual model and I kept the hyperparameters of the preprocessing functions at their default values.

After the optimization routine, all three model pipelines are passed to the `VotingClassifier` from `sklearn.ensemble` and are fitted with their incumbent configurations. Then, the imbalance sampling is removed from the pipeline in order to not sample for the test set. Thus, the final AutoML system is an ensemble of three pipelines with majority voting for the final classification.

3 Experiments

To run the optimization, I used a M1 Pro chip (2021) and 32 GB RAM. The external cross-validation performance in terms of balanced accuracy of the AutoML system vs. the untuned `RandomForestClassifer` baseline for each dataset id is shown below:

Model	976	980	1002	1018	1019	1021	1040	1053	1461	41160
Baseline	0.962	0.939	0.547	0.554	0.989	0.932	0.960	0.594	0.695	0.571
AutoML system	0.990	0.982	0.806	0.809	0.997	0.957	0.992	0.657	0.838	0.743
Improvement	0.027	0.043	0.259	0.254	0.008	0.025	0.032	0.063	0.143	0.172
McNemar test	37.19	30.94	927.63	906.28	8.75	11.40	1.86	287.88	660.02	117.14

Hence, the AutoML system outperforms the baseline across all datasets with an improvement ranging from 0.8% to 25.9%. To evaluate whether the two algorithms are significantly different, I used the McNemar test computed over the concatenated predictions of all folds. The reference distribution is χ_1^2, so the algorithms would be significantly different at the 5%-level if the test statistic is larger than 3.84, which is the case for all datasets except for dataset 1040. However, for this dataset the balanced accuracy is still 3.2% better and above 99%.

The plots of the trajectories for each dataset are shown in Appendix A. For the datasets 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 because SVMs are costlier to fit given the large number of observations and fewer iterations to tune the hyperparameters were possible. However, the overall performance of the AutoML system is still similar to the best individual pipelines due to majority voting. The benchmarks are usually outperformed after at most 50 seconds, except for the last two datasets due to the underperforming `SVC`. The final externally cross-validated performance is slightly lower than the performance of the individual algorithms for a few datasets, since the performance of the individual algorithms tends to be overly optimistic as the hyperparameters were tuned on that specific fold.

The selected incumbents for each model pipeline for all datasets are shown in Appendix B. Some 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 chosen. As expected, `SVC` uses the `StandardScaler` for all datasets, except 1019. If no sampling method was selected, the imbalance was always accounted for by class weights. Finally, the `RandomForestClassifier` tends to prefer deeper trees than the `GradientBoostingClassifier`.

4 Conclusion

The AutoML system outperforms the benchmark across all datasets and has thus demonstrated its usefulness. However, some improvements are still possible if a larger budget were available. The hyperparameters of the pre-processing methods could also be tuned to better fit to each dataset and algorithm. Furthermore, a stacking classifier could be trained on the three algorithms, which would allow to find the best weighted combination of the predictions of each individual algorithm for the final prediction of the AutoML system. I conducted experiments with stacking, but the training on the final incumbents with the highest budgets took fairly long, thus I decided to rather use the budget to improve the performance of the individual estimators. Overall, however, this AutoML system is capable of tuning well performing pipelines for imbalanced classification tasks.

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.

GeeksforGeeks (2023). Support vector machine in machine learning. `https://www.geeksforgeeks.org/support-vector-machine-in-machine-learning/`.

A Trajectories

The following plots show the trajectories of the incumbent performance of all three model pipelines (`RandomForestClassifier` (RFC), `RandomForestClassifier` (GBC) over runtime, `SVC`) evaluated with internal CV in each fold. Furthermore, the external CV of the untuned RFC baseline is plotted as grey horizontal line and the external CV of the final performance of the AutoML system is plotted as purple dot at the maximum runtime per model in each fold.

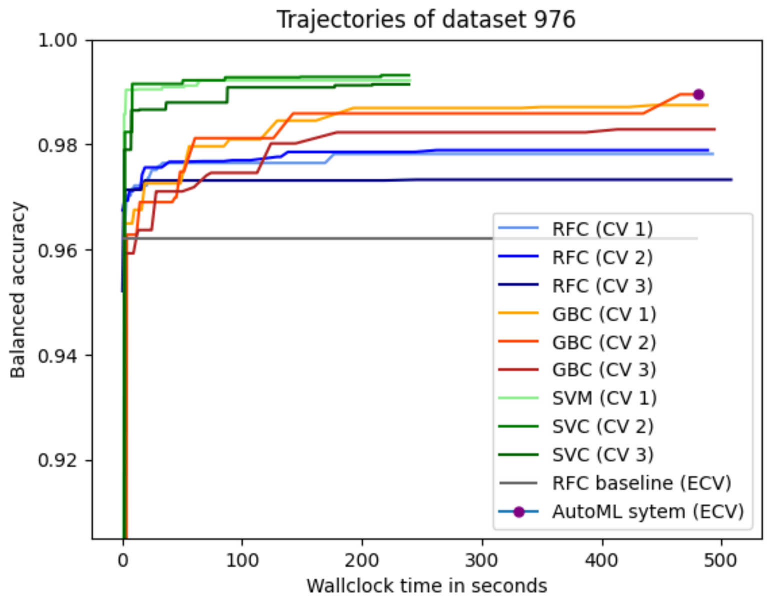

Figure 1: Trajectories for dataset 976 with 9961 observations and 14 features.

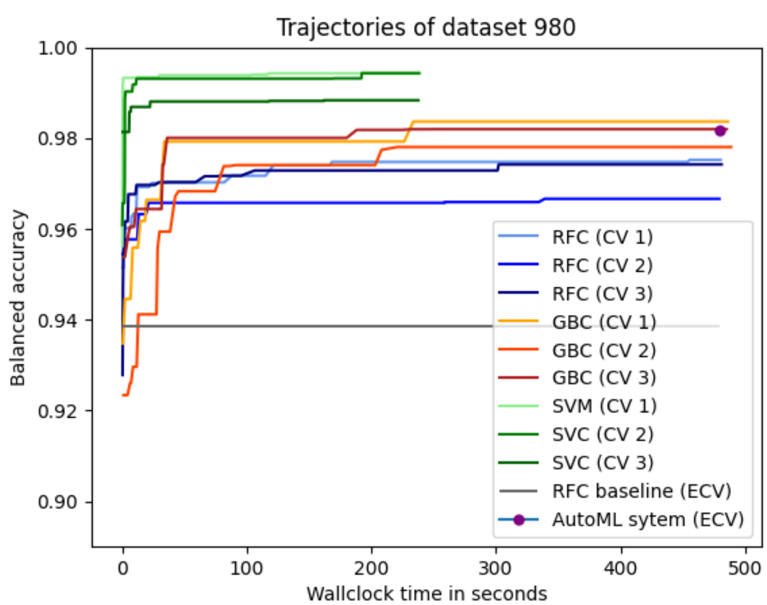

Figure 2: Trajectories for dataset 980 with 5620 observations and 64 features.

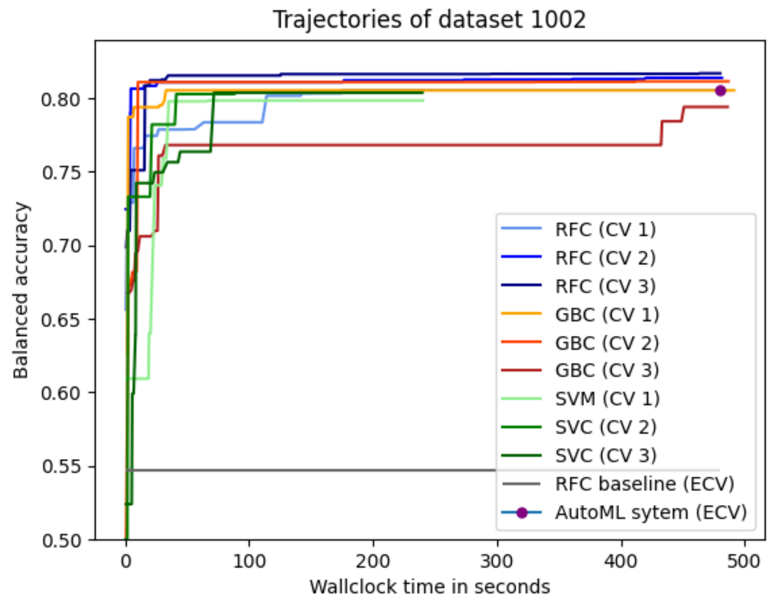

Figure 3: Trajectories for dataset 1002 with 7485 observations and 55 features.

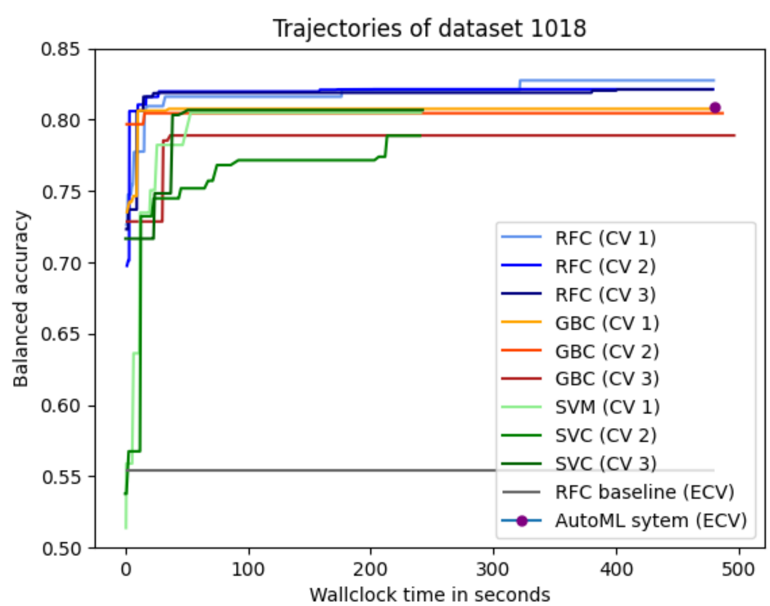

Figure 4: Trajectories for dataset 1018 with 8844 observations and 56 features.

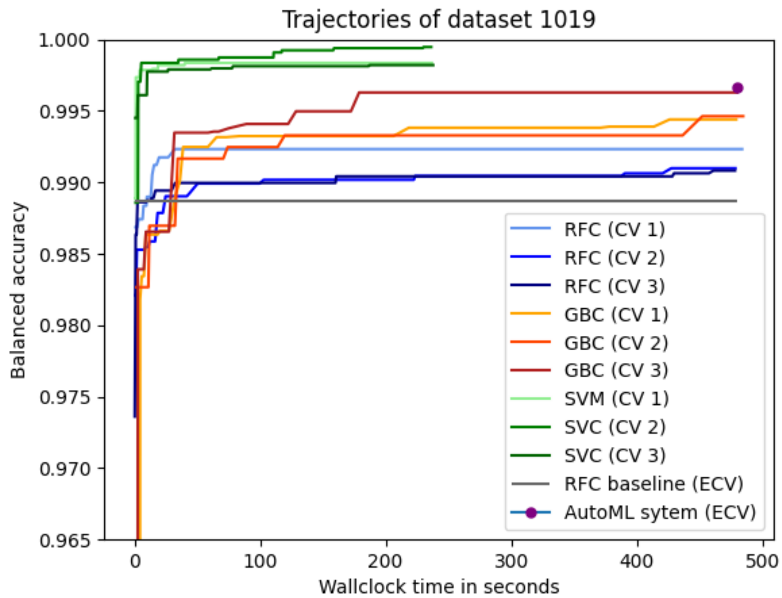

Figure 5: Trajectories for dataset 1019 with 10992 observations with 16 features

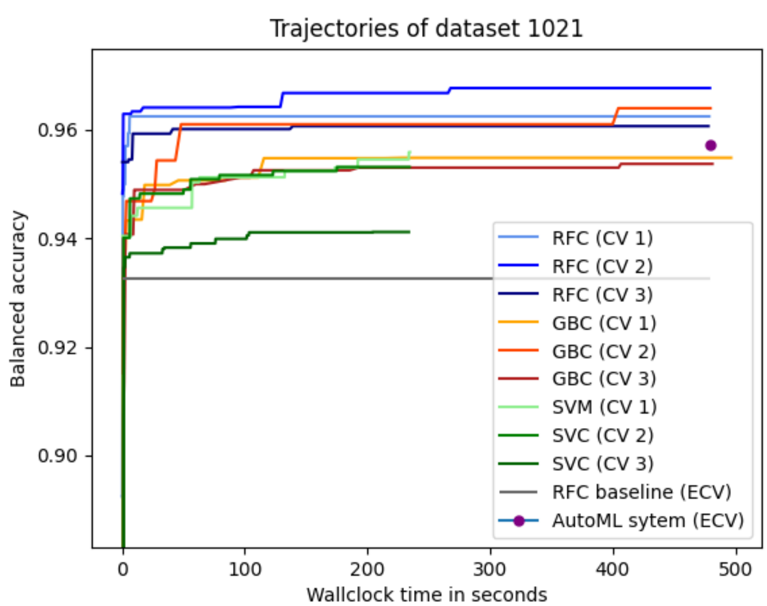

Figure 6: Trajectories for dataset 1021 with 5473 observations and 10 features.

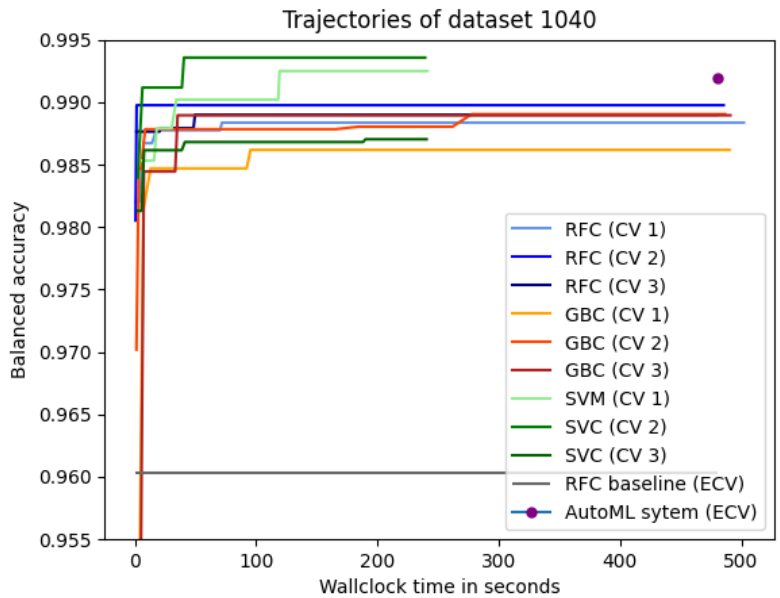

Figure 7: Trajectories for dataset 1040 with 14395 observations and 108 features.

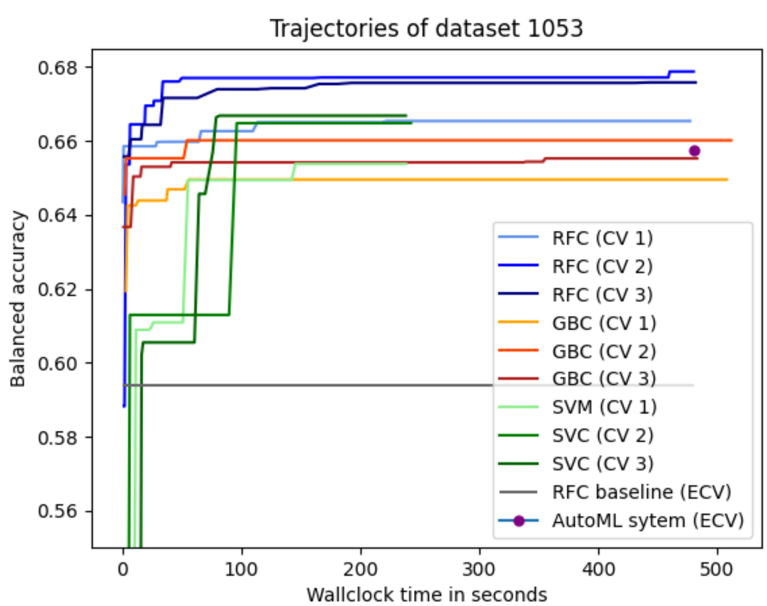

Figure 8: Trajectories for dataset 1053 with 10885 observations and 21 features.

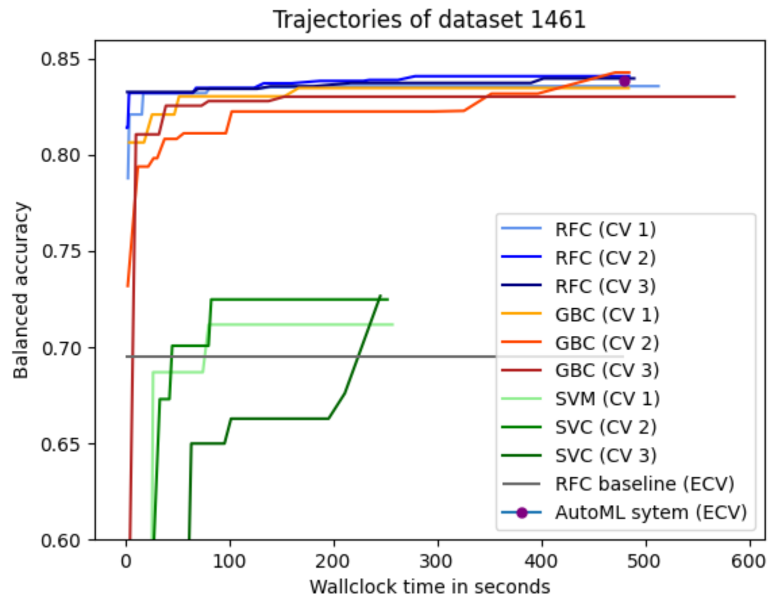

Figure 9: Trajectories for dataset 1461 with 45221 observations and 16 features.

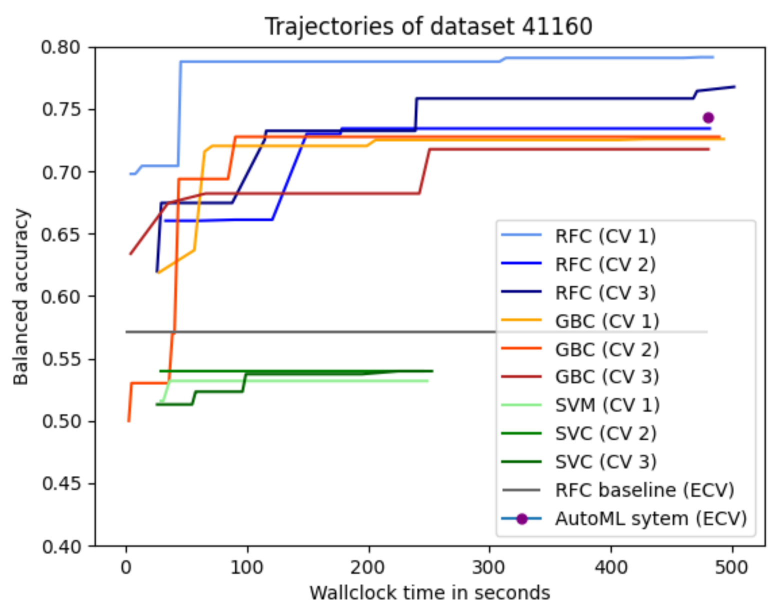

Figure 10: Trajectories for dataset 41660 with 31406 observations and 22 features.

B Incumbent hyperparameters

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

Table 1: Incumbent hyperparameters 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	True	entropy	19	6	1	0.527	balanced_subsample
976	2	Simple	SMOTETomek	True	log_loss	25	1	1	0.419	balanced_subsample
976	3	Simple	SMOTE	False	entropy	18	4	2	0.676	balanced
980	1	KNN	SMOTE	False	entropy	22	4	8	0.190	balanced_subsample
980	2	Simple	SMOTE	True	entropy	7	9	8	0.202	balanced_subsample
980	3	Simple	SMOTE	True	entropy	7	2	3	0.357	balanced
1002	1	KNN	Tomek	True	entropy	5	4	5	0.152	balanced
1002	2	KNN	SMOTE	True	entropy	6	13	1	0.106	None
1002	3	Simple	None	True	entropy	5	8	6	0.262	balanced
1018	1	KNN	Tomek	False	gini	5	10	1	0.188	balanced_subsample
1018	2	KNN	None	False	entropy	6	24	4	0.116	balanced
1018	3	Simple	Tomek	True	entropy	5	2	9	0.127	balanced_subsample
1019	1	KNN	SMOTETomek	True	entropy	17	4	2	0.664	balanced_subsample
1019	2	Simple	SMOTETomek	True	log_loss	21	3	3	0.295	None
1019	3	Simple	SMOTETomek	False	entropy	20	3	6	0.464	balanced
1021	1	KNN	SMOTETomek	True	gini	13	2	7	0.203	balanced
1021	2	Simple	SMOTETomek	False	gini	9	4	5	0.416	balanced
1021	3	KNN	SMOTE	False	entropy	14	4	9	0.568	balanced
1040	1	KNN	Tomek	True	log_loss	12	1	15	0.513	balanced
1040	2	KNN	SMOTE	True	log_loss	6	32	3	0.285	balanced_subsample
1040	3	Simple	None	True	log_loss	17	1	5	0.365	balanced_subsample
1053	1	KNN	Tomek	False	log_loss	8	10	16	0.590	balanced_subsample
1053	2	Simple	Tomek	True	gini	12	24	4	0.400	balanced
1053	3	KNN	Tomek	False	entropy	25	9	14	0.555	balanced
1461	1	Simple	None	False	entropy	11	5	16	0.628	balanced
1461	2	Simple	None	False	gini	9	10	1	0.390	balanced
1461	3	KNN	Tomek	False	entropy	15	8	15	0.765	balanced_subsample
41160	1	Simple	Tomek	False	log_loss	12	28	11	0.811	balanced
41160	2	Simple	Tomek	False	gini	10	3	5	0.379	balanced
41160	3	KNN	None	True	gini	12	5	9	0.851	balanced

Table 2: Incumbent hyperparameters 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	False	exponential	0.603	squared_error	2	7	6
976	2	Simple	SMOTETomek	True	log_loss	0.522	squared_error	23	3	5
976	3	Simple	SMOTETomek	True	exponential	0.815	squared_error	5	3	3
980	1	Simple	SMOTE	True	log_loss	0.598	squared_error	7	4	4
980	2	KNN	SMOTETomek	True	log_loss	0.432	friedman_mse	9	7	3
980	3	Simple	SMOTETomek	True	exponential	0.322	squared_error	6	1	5
1002	1	KNN	SMOTETomek	True	log_loss	0.010	squared_error	3	7	2
1002	2	KNN	SMOTETomek	False	log_loss	0.013	squared_error	7	6	2
1002	3	KNN	SMOTE	True	exponential	0.021	friedman_mse	12	7	4
1018	1	KNN	SMOTETomek	True	log_loss	0.061	friedman_mse	6	1	3
1018	2	Simple	SMOTETomek	True	log_loss	0.185	squared_error	6	12	3
1018	3	KNN	SMOTE	True	exponential	0.054	friedman_mse	2	1	3
1019	1	Simple	SMOTE	True	exponential	0.398	squared_error	31	4	4
1019	2	Simple	SMOTETomek	True	log_loss	0.158	friedman_mse	7	1	3
1019	3	Simple	SMOTE	False	exponential	0.666	friedman_mse	5	6	4
1021	1	KNN	SMOTETomek	True	exponential	0.018	friedman_mse	7	6	5
1021	2	Simple	SMOTE	False	log_loss	0.034	squared_error	3	4	5
1021	3	KNN	SMOTETomek	True	log_loss	0.141	friedman_mse	7	3	6
1040	1	KNN	SMOTETomek	True	log_loss	0.048	friedman_mse	5	3	4
1040	2	Simple	SMOTE	False	exponential	0.588	friedman_mse	4	3	3
1040	3	Simple	SMOTE	False	log_loss	0.410	friedman_mse	20	7	3
1053	1	KNN	SMOTETomek	True	exponential	0.014	squared_error	15	1	3
1053	2	Simple	SMOTETomek	False	exponential	0.021	squared_error	7	3	3
1053	3	KNN	SMOTETomek	True	log_loss	0.270	squared_error	8	8	2
1461	1	KNN	SMOTETomek	False	exponential	0.092	squared_error	7	4	7
1461	2	KNN	SMOTETomek	False	log_loss	0.035	friedman_mse	4	7	5
1461	3	Simple	SMOTETomek	False	log_loss	0.111	friedman_mse	8	5	8
41160	1	Simple	SMOTE	False	log_loss	0.023	friedman_mse	16	2	11
41160	2	Simple	SMOTE	False	exponential	0.043	friedman_mse	30	2	9
41160	3	Simple	SMOTETomek	False	log_loss	0.058	squared_error	5	3	11

Table 3: Incumbent hyperparameters configurations for the SVM classifier

Dataset ID	CV fold	Imputer	Sampler	Scaler	C	Kernel	Shrinking	Tolerance	Class weight
976	1	KNN	SMOTETomek	True	8.489	rbf	True	0.0085	balanced
976	2	Simple	None	True	8.670	rbf	False	0.0016	balanced
976	3	Simple	None	True	4.036	rbf	False	0.0005	balanced
980	1	Simple	SMOTETomek	True	2.230	poly	False	0.0040	balanced
980	2	KNN	Tomek	True	0.893	poly	True	0.0030	balanced
980	3	KNN	None	False	1.061	rbf	True	0.0006	balanced
1002	1	KNN	SMOTETomek	True	0.161	linear	True	0.0017	balanced
1002	2	KNN	Tomek	True	0.199	linear	False	0.0003	balanced
1002	3	Simple	SMOTE	True	0.169	linear	False	0.0025	balanced
1018	1	Simple	SMOTETomek	True	0.213	sigmoid	True	0.0007	None
1018	2	Simple	Tomek	True	0.621	poly	True	0.0001	balanced
1018	3	KNN	SMOTETomek	True	0.113	linear	True	0.0039	balanced
1019	1	Simple	None	False	2.186	poly	True	0.0062	balanced
1019	2	Simple	SMOTETomek	False	6.871	rbf	True	0.0040	None
1019	3	Simple	Tomek	False	7.892	rbf	True	0.0003	balanced
1021	1	Simple	None	True	6.803	rbf	True	0.0058	balanced
1021	2	Simple	Tomek	True	7.117	rbf	True	0.0011	balanced
1021	3	Simple	Tomek	True	6.244	rbf	True	0.0005	balanced
1040	1	KNN	SMOTE	True	0.218	linear	True	0.0019	None
1040	2	KNN	SMOTETomek	True	0.143	linear	True	0.0001	balanced
1040	3	KNN	SMOTE	True	1.384	sigmoid	False	0.0012	balanced
1053	1	Simple	None	True	0.851	rbf	True	0.0009	balanced
1053	2	Simple	SMOTE	True	0.384	rbf	True	0.0009	balanced
1053	3	Simple	Tomek	True	0.900	rbf	False	0.0012	balanced
1461	1	Simple	None	True	0.363	sigmoid	False	0.0004	balanced
1461	2	KNN	Tomek	True	0.399	sigmoid	False	0.0004	balanced
1461	3	Simple	SMOTETomek	True	0.526	rbf	False	0.0003	balanced
41160	1	Simple	SMOTETomek	True	0.190	sigmoid	True	0.0002	balanced
41160	2	KNN	SMOTE	True	0.586	sigmoid	True	0.0007	balanced
41160	3	KNN	SMOTETomek	True	1.275	rbf	False	0.0037	None

C Meta data of datasets

Table 4: Meta data of each dataset

Dataset ID	Observations	Features	Share of underrepresented class	Total missing values
976	9961	14	0.162	0
980	5620	64	0.102	0
1002	7485	55	0.106	0
1018	8844	56	0.064	0
1019	10992	16	0.104	0
1021	5473	10	0.102	0
1040	14395	108	0.062	0
1053	10885	21	0.193	25
1461	45211	16	0.117	0
41160	31406	22	0.095	29756