Example Submission Final Project AutoML Lecture WS 2022/2023

Constantin von Crailsheim[1]

[1]LMU Munich, Institute of Statistics

Abstract

1 Introduction

The AutoML system optimizes a ML pipeline consisting of four elements. First, a choice of two imputers replaces missing values. Then, either over- or undersampling was applied to deal with imbalance in the targets. After 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. Finally, the best pipelines for each model were stacked and the final AutoML system predicts the target using majority voting.

2 Method

Choices of imputation strategy of sklearn.impute:

- 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 and will most likely still produce sensible imputed values for categorical features.
- The KNNImputer replaces missing values by the mean value for that feature of its 5 nearest neighbors based on Euclidian distance of their non-missing feature observations.

Choices of data-level sampling method to yield balanced dataset of imblearn:

- SMOTE as an oversampling approach generates new samples of the minority class by interpolating between existing observations of the minority class, while not distinguishing 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 examples 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.

Since the above pre-processing methods impute missing values by means or medians and generate new samples by interpolation, values for categorical features could be generated which are not actually categorical anymore. Thus, another layer is added to the pipeline, which rounds all observations of categorical 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 a RBF kernel assumes features centered around zero and similar variance across features.

Subsequently, a the hyperparameter for the three models of the stack were optimized, where the search space was defined with the ConfigSpace package. For almost all hyperparameter, the default was set to the default of the 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 stack 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 to be optimized, which will all be sampled uniform, are as following:

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	

Comment on hyperparameter

The second model in the stack is a GradientBoostingClassifier from sklearn.ensemble. Why? The hyperparameters to be optimized are as following:

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	

Comment on hyperparameter

The third model in the stack is a Support Vector Classifier (SVC) from sklearn.svm. Why? The hyperparameters to be optimized are as following:

Hyperparameter	Data type	Search space	Default	Other
C	Float	[0.1, 10]	1.0	Log scale
kernel	Categorical	{Linear, Polynomial, RBF, Sigmoid}	RBF	
shrinking	Boolan	{True, False}	True	
tol	Float	[1e-5,1e-3]	1e-3	
class_weight	Categorical	{Balanced, None}	None	

Comment on hyperparameter

To optimize the hyperparameter of the AutoML system, DEHB by Awad et al. (2021) was used, which combines Differential Evolution and Hyperband. Differential evolution generates a new mutant vector from three random parents and then generates the offspring by random selecting values from the new mutant vector with probability p and from on of the corresponding parents otherwise. Hyperband allows to search the whole search space with cheap evaluations and only train more expensive models on promising areas of the search space. The algorithm starts by sampling $N = \eta^{f-1}$ random hyperparameter configurations, which are evaluated at the lowest budget. Then the best $1/\eta$ of the configurations are evaluated at a η times as higher fidelity and this process is repeated until the highest fidelity (denoted here by f) is reached. After completing one iteration, the algorithm restarts with new instantiations, but evaluating these at the second lowest fidelity. DEHB combines both approaches by generating the configurations for the next fidelity by differential evolution from the lower fidelity as parent pool, where the previous evaluations indicate promising regions. 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 we have for various hyperparameter. Their experiments have shown that DEHB also outperforms SMAC by mean ranks across all benchmarks. For those reasons, DEHB was chosen as an efficient optimizer. Key features of DEHB:

For the optimization, $\eta = 3$ and $f = 4$, which implies an initial population of $N = 3^3 = 27$. The budget for the RandomForestClassifier and the GradientBoostingClassifier, as indicated by the number of trees in the forest, was set to a minimum of 20 and maximum of 540. For SVC, the budget is indicated by the maximum number of iterations and was set to a minimum of 500 and maximum of 13500. However, the runtime between the lowest and largest budget did not differ too much and was usually around 0.2 to 1.0 seconds. Thus, the SVC mostly benefits from the differential evolution and the successive halving element is not as important since many configurations can be tested irrespecitively. The evaluations for the RandomForestClassifier and GradientBoostingClassifier, the evaluations at higher fidelities are more costly (How costly?) Thus, 40% of the maximum cost was allocated to optimizing these models, and 20% was allocated to optimizing the SVC.

To evaluate the performance of the AutoML system, 3-fold external and 4-fold internal cross-validation was used. 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, with 480 second allocated to the tree models and 240 seconds to the SVC. To evaluate each hyperparameter configuration, 4-fold cross-validation was used. Since the budget per hyperparameter configuration is not that big after accounting for cross-validation, only a selection of hyperparameters of the actual model were optimized and the hyperparameters of the preprocessing functions were kept at their default values.

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

3 Experiments

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

The chosen incumbents for each model for dataset x are shown below:

Example of trajectory for dataset x.

Model	976	1002
Baseline	0.960	0.537
AutoML system	0.985	0.810
Improvement	0.025	0.273

4 Conclusion

Improvements: Also optimize hyperparameter of pre-processing methods.

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

Bla Bla

B Incumbent hyperparameter