General constraints for code submissions Please adhere to these rules to make our and your life easier! We will deduct points if your solution does not fulfill the following:

- If not stated otherwise, we will use exclusively Python 3.7.
- If not stated otherwise, we expect a Python script, which we will invoke exactly as stated on the exercise sheet.
- Your solution exactly returns the required output (neither less nor more) – you can implement a `--verbose` option to increase the verbosity level for developing.
- Add comments and docstrings, so we can understand your solution.
- (If applicable) The `README` describes how to install requirements or provides addition information.
- (If applicable) Add required additional packages to `requirements.txt`. Explain in your `README` what this package does, why you use that package and provide a link to it's documentation or GitHub page.
- (If applicable) All prepared unittests have to pass.
- (If applicable) You can (and sometimes have to) reuse code from previous exercises.

We have learned all the components for building a first AutoML optimizer. Now we want to use these components to build a full AutoML pipeline and win an internal competition. In this competition, we will provide three training datasets to you where you can evaluate your AutoML models locally. Another 2 unseen datasets will be provided as test sets – of course, you do not have access to these. The team with the best score on the test sets will win the competition. You can use whatever techniques you have implemented in this lab course previously: Bayesian optimization, evolutionary algorithms, meta-learning, etc. Of course, you can also add your own ideas. However, your model should be built based on your own implementation, i.e., you can not use any of the existing AutoML packages (e.g. autosklearn, skopt, opentuner, nevergrad, etc.) to win the competition.

Basically, your model should be trained with a training-validation set (you can freely split into training and validation set) to determine a well-performing model and its configuration in the given time. The finally returned hyperparameter configuration is then fitted to the training data and evaluated on the test set. Your system will be given 20 minutes to be optimized on each dataset − you need to be efficient!

Both training and test datasets are binary classification problems only. There is no missing data; no categorical features in the datasets. However, the number of features and instances might differ.

The score is computed with balanced accuracy (`BAC`), which is the average of `sensitivity (true positive rate)` and `specificity (true negative rate)`:

$$BAC = \frac{1}{2}\left[\frac{TP}{P} + \frac{TN}{N}\right] \tag{1}$$

where $P(N)$ is the number of positive (negative) examples, $TP(TN)$ is the number of correctly classified positive (negative) examples. Then `BAC` will be normalized with:

$$|BAC| = (BAC - R)/(1 - R) \tag{2}$$

where $R = 0.5$ for our binary classifier problem.

In general, you need to implement the following pieces:

1. `Configuration Space`: `sklearn_configspace.py` provides several configuration spaces for different `sklearn` models. One idea could be to need to merge them into a single configuration space to let your optimizer have more choices.
2. `Run History`: you need to implement `RunHistory` in `util.py` by yourself, notice that this time runhistroy might not store a simple list as the previous exercises sections.
3. `Initialization`: you need to initialize your configurations (and fidelities).
4. `Optimization loops`: you need to implement the main optimization loop of your AutoML system. Once the overall budget is exhausted, the optimizer will be forced to terminated
5. You can add whatever you want to boost your performance.

Please upload your implementation to your GitHub repository before the deadline. We will only evaluate your last submission before the deadline, any submissions after that will be ignored.