Project 5: ML and the Garden of Forking Paths with mlr3 and mlr3pipelines

This repository contains the code, related data and documentation for conducting a benchmark analysis on different learners with different evaluation techniques using different pre-processing pipelines with the use of mlr3 and mlr3pipelines packages in R.

Background

Benchmark analyses are often used as a way to evaluate the performance of different models or algorithms. However, these benchmark experiments can be affected by many choices made by the researcher, such as how the data is pre-processed and which evaluation measure is used. These choices, known as "researcher degrees of freedom", can influence the results of the experiment.

"Researcher degrees of freedom" refers to the flexibility and discretion that researchers have when conducting a benchmark analysis. This flexibility can be particularly high when pre-processing the data and deciding how to evaluate the results.

In the context of pre-processing the data, the researcher has discretion in how to clean, transform, or manipulate the data to make it more suitable for machine learning. Depending on the researcher's choices, this can significantly impact the performance of the machine learning model.

When evaluating the results, the researcher also has discretion in the choice of evaluation measure. Depending on the choice of evaluation measure, the performance of the model can appear differently.

The **goal** of this project is to highlight that depending on the pipe-line and evaluation measure, different algorithms could be shown to have superior performance.

Requirements and Dependencies

This code is written in R and requires the following software to be installed:

- R- 4.2.2
- RStudio- Latest version (2022.12.0-353 for Windows, 2022.12.0-353-amd64.deb for Ubuntu)

The following R packages are required to run the code:

- mlr3
- mlr3pipelines
- mlr3verse
- mlr3learners
- paradox
- mlr3tuning
- mlr3viz
- mlr3tuningspaces
- mlr3oml
- ggplot2
- dplyr
- gridExtra

- igraph
- ranger
- xgboost
- R6

To install the required packages, you can use the following command in RStudio:

install.packages('<package_name>')

Code Structure

The entire code is divided into **6 sections**.

- **Section 1:** In this section, all the necessary libraries are installed and imported, the tasks are set and the AutoTuner class configurations are defined. These configurations include the hyperparameter tuning method that is to be used (for example: Grid Search, Bayesian Optimization, Random Search etc. Grid Search has been used as the tuner in this code), the terminator which sets when the tuning should stop (for example: Stopping the tuning after a certain period of time), the resampling method (for example: Cross Validation with 3 folds)

- **Section 2:** In this section the learners (classifiers), the hyperparameter search space for these learners, and the pre-processing pipelines are defined. First, pre-processing pipelines are created by using the mlr3pipelines library. A more in-depth explanation of the library is given in the published book. The pipelines are created using PipeOps (Pipeline Operators). Each PipeOps has a specific function. To

pre-process the data, the mean imputation and scaling operators are used. After defining various pipelines for each learner, the AutoTuner instances are defined for each learner where the AutoTuner class configurations defined earlier are used. The learners are then benchmarked and their performance on different tasks are recorded with respect to different evaluation measures. Based on this, these learners are then ranked.

- **Section 3:** The pre-processing pipeline is changed by using Histogram Imputation instead of Mean Imputation. The hyperparameter search space and all other AutoTuner class configurations are kept same. The learners are then benchmarked and all evaluations measures are again kept same. The rank and classification error results are aggregated.

- **Section 4:** The pre-processing pipeline is again changed by using a Variance Filter that removes features having noise (low variance). Since missing values in the dataset must be filled, this filter is combined with Mean Imputation and Scaling PipeOps. Everything else is kept the same.

- **Section 5:** In this section, the hyperparameters are not tuned and are set to arbitrary values to see if there is a change in performance. The pre-processing pipeline is kept same as the one used in Section 2.

- **Section 6:** After evaluating all the pipelines with different learners and evaluation scores on different datasets, the results are visualized. The Classification Error, Brier Score and AUC Scores are

aggregated into a visual matrix format for all pipelines applied on all datasets. Such a visualization gives a compact and easier comparison of results. The same has been done for the rankings of the learners: For each evaluation measure, the change in rankings is seen with change in the pre-processing pipeline applied on different datasets.

Running the Code

The code has been tested on Linux (Ubuntu) and Windows operating systems.

Ensure that R has been installed on your system. If not, please install it from: https://cran.r-project.org/

When using Linux and if using RStudio, write the following commands onto the terminal to ensure that the packages are installed properly:

```
sudo apt-get install r-base-dev
sudo apt-get install libcurl4-openssl-dev libxml2-dev
```

During benchmarking in Section 2, Section 3, Section 4, especially during the Hyperparameter Optimization phase, the code takes quite some time to run. Each benchmarking session from Section 2 to 4 may take 15-20 minutes. Benchmarking in Section 5 does not tune the hyperparameters, hence it executes relatively faster.

An important thing to keep in mind is that during benchmarking, the terminal may remain blank and it may

seem that the terminal has frozen but the code is still running in background.

It is recommended to execute one code line at a time and not the whole code all at once. For example: It is recommended to move to the next line after benchmarking only after the entire benchmarking process has finished.

The results should be **reproducible** in the sense that on running the code, one should notice the change in performance with respect to different pipelines and evaluation measures, as aggregated in the plot images.

Example: For Classification Error [Refer to Image: Rankings_CE.png]

- **PIMA Dataset**: Change in performance with respect to pipeline is seen in the case of 'notune' pipeline where no tuning of the hyperparameters is done. In this case, Decision Tree seems to perform better than Random Forest.
- **Sonar Dataset:** Again, 'notune' pipeline shows change in rankings. Decision Tree performs the best out of all learners in the case of arbitrary hyperparameter assignment. Otherwise, the rankings remain unchanged.
- **Blood Transfusion Service Center Dataset:** Because of the variance filter pre-processing step, Decision tree and Random Forest's ranking falls.
- **Breast Cancer:** For all 4 pipelines, Decision Tree and Featureless classifier's performance doesn't change. But the performance of Random Forest gets better when Mean Imputation pipeline is used.

Various images have been included in the 'Results' folder to showcase how the performance of different learners on different datasets is changing depending on what evaluation measure and which pipeline is used.

The change in rankings of learners is also showcased and it can be seen that due to changes in pipelines or changing the hyperparameter to arbitrary values, the performance of the learners' changes. In a particular pipeline, a learner may be inferior with respect to another learner and in another pipeline, the very same learner may even be superior, hence this is the **result that is to be noted**: that depending on the choice of evaluation measure, the pre-processing method, the performance of the learner can appear differently.