# fseval: A Benchmarking Framework for Feature Selection and Feature Ranking Algorithms

Jeroen G. S. Overschie [1], **Ahmad Alsahaf** [2], **and George Azzopardi** [1]

1 Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands **2** Department of Biomedical Sciences of Cells and Systems, University Medical Center Groningen, University of Groningen, 9713 GZ Groningen, The Netherlands

Summary

The `fseval` Python package allows benchmarking Feature Selection and Feature Ranking algorithms on a large scale, and facilitates the comparison of multiple algorithms in a systematic way. In particular, `fseval` enables users to run experiments in parallel and distributed over multiple machines, and export the results to an SQL database. The execution of an experiment can be fully determined by a configuration file, which means the experiment results can be reproduced easily, given only the configuration file. `fseval` has high test coverage, continuous integration, and rich documentation. The package is open source and can be installed through PyPI. The source code is available at: https://github.com/dunnkers/fseval.

Statement of Need

Feature Selection (FS) and Feature Ranking (FR) are extensively researched topics within machine learning (Guyon & Elisseeff, 2003; Venkatesh & Anuradha, 2019). FS methods determine subsets of relevant features in a dataset, whereas FR methods rank the features in a dataset relative to each other in terms of their relevance. When a new FS or FR method is developed, a benchmarking scheme is necessary to empirically validate its effectiveness. Often, the benchmark is conducted as follows: features are ranked by importance, then the predictive quality of the feature subsets containing the top ranked features is evaluated using a validation estimator. Some studies let the competing FS or FR algorithms pick out a fixed number of top k features and validate the performance of that feature subset (Bradley & Mangasarian, 1998; Roffo et al., 2015; Zhao & Liu, 2007), whilst others evaluate multiple subsets of increasing cardinality containing the highest ranked features (Almuallim & Dietterich, 1991; Bennasar et al., 2015; Gu et al., 2012; Kira & Rendell, 1992; Peng et al., 2005; Wojtas & Chen, 2020). FS algorithms that only make a binary prediction on which features to keep, are always evaluated in the former way.

There is a clear case for performing Feature Selection, as it has been shown to improve classification performance in many tasks, especially those with a large number of features and limited observations. In those applications, it is difficult to determine which FS method is suitable in the general case. Therefore, large empirical comparisons of several FS methods and classifiers are routinely performed. For instance, in microarray data (Cilia et al., 2019), medical imaging (Sun et al., 2019; Tohka et al., 2016), and text classification (Kou et al., 2020; Liu et al., 2017). Therefore, it is valuable to find out emperically which FR- or FS method works best. This requires running a benchmark to do so.

`fseval` is an open-source Python package that helps researchers perform such benchmarks efficiently by eliminating the need for implementing benchmarking pipelines from scratch to

test new methods. The pipeline only requires a well-defined configuration file to run - the rest of the pipeline is automatically executed. Because the entire experiment setup is deterministic and captured in a configuration file, results of any experiment can be reproduced given the configuration file. This can be very convenient to researchers in order to prove the integrity of their benchmarks.

To the best of our knowledge, there is only one package that aims to accomplish a similar goal (`featsel`, (Reis et al., 2017)). Compared to this tool, `fseval` is easier to install and use, has better documentation, and is better maintained. `fseval` also has more extensive functionalities compared to `featsel`: with support for easily configurable and reproducible pipeline configuration using either YAML or Python and distributed-processing support. Due to the lack of functionality and the fact that the refered-to library is out-of-date, we consider there to be a gap in the field, which our library aims to fill.

- The **target audiences** are researchers in the domains of Feature Selection and Feature Ranking, as well as businesses that are looking for the best FR- or FS method to use for their use case.
- The **scope** of `fseval` is limited to handle tabular datasets for the classification and regression objectives.

Key Features

`fseval` is a flexible and unbiased framework which provides as much useful functionality as possible. Most features are optional, and can be enabled or disabled according to what the user deems fit. The aim of the package is to accommodate the most common benchmarking settings and protocols that feature selection researchers use.

- **Algorithm support**. FR or FS algorithms that estimate the importance of features in various ways are supported, including the following output attributes: (1) a FEATURE_IMPORTANCES_ vector in $\mathbb{R}$, (2) a RANKING_ vector in $\mathbb{Z}$ and (3) a SUPPORT_ vector in $\mathbb{B}$. An estimator might support any combination of the output attributes. Once estimators are fit there is the option to save a *cached* version.

- **Dataset** adapters. Datasets can be loaded from multiple sources using *adapters*. Users can implement adapters themselves by implementing a given interface, or use a built-in adapter class to load datasets from OpenML (Vanschoren et al., 2013). Adapters might also be functions, which, for example, allow users to directly use the sklearn functions `make_classification` or `make_regression` as adapters to create **synthetic** datasets. Datasets might also define dataset feature importance *ground truths*, which can be used to compute metrics in the scoring stage (Section 'Scoring').

- **Built-in integrations**. `fseval` allows exporting benchmark results directly to various SQL databases using SQLAlchemy (Bayer, 2012), or to the Weights and Biases experiment tracker platform (Biewald, 2020). Users can create custom metrics and perform aggregations over the bootstrap results.

- **Scalable and distributed** computing. Besides that the process of running multiple bootstraps can be distributed over the CPU, `fseval` also allows executing experiments on SLURM clusters (Yoo et al., 2003) or on the cloud platform AWS. This is possible because all configuration regarding the execution of the pipeline can be captured in a configuration file.

- **Reproducible** experiments. Because the entire execution state of the pipeline can be expressed in a single configuration file, it is easy to reproduce experiment results. Given that a scientist uses estimates that are deterministic (e.g. by fixing a RANDOM_STATE variable), others can reproduce the results, improving the scientific integrity of the work.

The Pipeline

`fseval` executes a predefined sequence of steps, as can be seen in Figure 1.
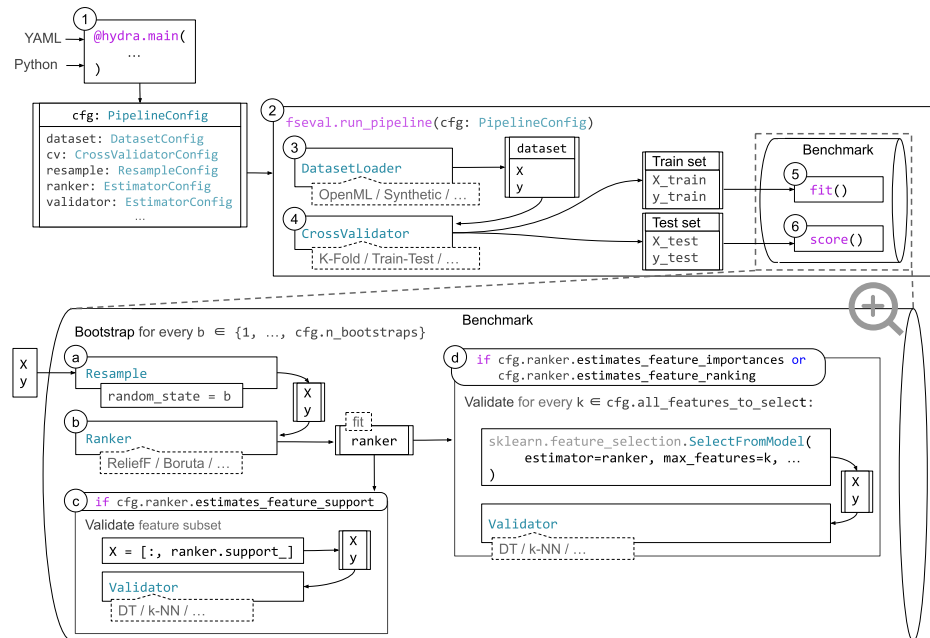
Figure 1: A schematic of the benchmarking pipeline. The input of the pipeline is at all times a `PipelineConfig` object, processed from YAML or Python by Hydra. After steps 1-6, steps a-d are executed for both the fitting step and scoring step.

First, in step 1, the pipeline configuration is processed using Hydra (Yadan, 2019). Hydra is a powerful tool for creating Command Line Interfaces in Python, allowing hierarchical representation of the configuration. Configuration can be defined in either YAML or Python files, or a combination of the two. The top-level config is enforced to be of the `PipelineConfig` interface, allowing Hydra to perform type-checking. The config is then passed to the `run_pipeline` function in step 2. Then, after the dataset is loaded in step 3, the splits for cross validation are determined in step 4. Each cross validation fold is executed in a separate run of the pipeline. The training and testing subsets are then given to the *fitting* and *scoring* steps, steps 5 and 6, respectively.

Fitting

In the fitting step, the Feature- Ranker or Selector and validation estimators are fit on the given training set. The validation estimator is fit on all feature subsets that are desired to be evaluated. For every bootstrap $b \in \{1, \dots, \texttt{PipelineConfig.n\_bootstraps}\}$, a fit sequence is run. The bootstraps can be distributed over CPUs by setting `PipelineConfig.n_jobs` > 1. The fit process consists of the following steps.

(a) The dataset is resampled according to the `PipelineConfig.resample` config, using `random_state` $= b$.

(b) The FR or FS algorithm is fit. Then, the estimator can be cached as a pickle file.

(c) If the ranker estimates SUPPORT_ (*Feature Selection*): The selected feature subset is validated using the validation estimator.

(d) If the ranker estimates FEATURE_IMPORTANCES_ or RANKING_ (*Feature Ranking*) then

every number in the list `PipelineConfig.all_features_to_select` is used to take the k best features in the ranking, and fitting the validation estimator on the subset.

Scoring

After the estimators have been fit, a scoring step is executed on the test set. By default, the validation estimator score function is triggered and its results are stored. Depending on the estimator, this often means *classification accuracy* for classifiers and the R^2 score for regressors. Besides the built-in metrics, users can install custom metrics.

To install custom metrics, programmatic hooks are available. This enables, for example, a user aggregate over the various validated feature subsets and bootstrapped datasets. A user could compute the average accuracy over all bootstraps, or compute various stability metrics (Nogueira et al., 2018). Another example of a custom metric would be to compare the dataset ground-truth feature importances to the estimated importances, which information would be available when using *synthetic* datasets.

Conclusion and Future Work

`fseval` is a comprehensive and feature rich Python library for benchmarking Feature Ranking and Feature Selection algorithms. It allows authors to focus on their empirical research instead of having to implement another benchmarking pipeline - exploiting `fseval`'s support for parallel processing, distributed computing and export possibilities. `fseval` is open source and published on the PyPi platform. Next steps are to include more built-in dataset adapters, metrics and export possibilities.

References

Almuallim, H., & Dietterich, T. G. (1991). Learning With Many Irrelevant Features. *In Proceedings of the Ninth National Conference on Artificial Intelligence*, 547–552.

Bayer, M. (2012). SQLAlchemy. In A. Brown & G. Wilson (Eds.), *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org. http://aosabook.org/en/sqlalchemy.html

Bennasar, M., Hicks, Y., & Setchi, R. (2015). Feature selection using Joint Mutual Information Maximisation. *Expert Systems with Applications*, *42*(22), 8520–8532. https://doi.org/10.1016/j.eswa.2015.07.007

Biewald, L. (2020). *Experiment Tracking with Weights and Biases*. https://www.wandb.com/

Bradley, P. S., & Mangasarian, O. L. (1998). Feature Selection via Concave Minimization and Support Vector Machines. *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98*, 82–90.

Cilia, N. D., De Stefano, C., Fontanella, F., Raimondo, S., & Scotto di Freca, A. (2019). An experimental comparison of feature-selection and classification methods for microarray datasets. *Information*, *10*(3). https://doi.org/10.3390/info10030109

Gu, Q., Li, Z., & Han, J. (2012). Generalized Fisher Score for Feature Selection. *arXiv:1202.3725 [Cs, Stat]*. http://arxiv.org/abs/1202.3725

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *The Journal of Machine Learning Research*, *3*(null), 1157–1182.

Kira, K., & Rendell, L. (1992). The Feature Selection Problem: Traditional Methods and a New Algorithm. *AAAI*.

Kou, G., Yang, P., Peng, Y., Xiao, F., Chen, Y., & Alsaadi, F. E. (2020). Evaluation of feature selection methods for text classification with small datasets using multiple criteria decision-making methods. *Applied Soft Computing*, *86*, 105836. https://doi.org/10.1016/j.asoc.2019.105836

Liu, Y., Bi, J.-W., & Fan, Z.-P. (2017). Multi-class sentiment classification: The experimental comparisons of feature selection and machine learning algorithms. *Expert Systems with Applications*, *80*, 323–339. https://doi.org/10.1016/j.eswa.2017.03.042

Nogueira, S., Sechidis, K., & Brown, G. (2018). On the Stability of Feature Selection Algorithms. *Journal of Machine Learning Research*, *18*(174), 1–54. http://jmlr.org/papers/v18/17-514.html

Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(8), 1226–1238. https://doi.org/10.1109/TPAMI.2005.159

Reis, M. S., Estrela, G., Ferreira, C. E., & Barrera, J. (2017). Featsel: A framework for benchmarking of feature selection algorithms and cost functions. *SoftwareX*, *6*, 193–197. https://doi.org/10.1016/j.softx.2017.07.005

Roffo, G., Melzi, S., & Cristani, M. (2015). Infinite Feature Selection. *2015 IEEE International Conference on Computer Vision (ICCV)*, 4202–4210. https://doi.org/10.1109/ICCV.2015.478

Sun, P., Wang, D., Mok, V. C., & Shi, L. (2019). Comparison of feature selection methods and machine learning classifiers for radiomics analysis in glioma grading. *IEEE Access*, *7*, 102010–102020. https://doi.org/10.1109/ACCESS.2019.2928975

Tohka, J., Moradi, E., & Huttunen, H. (2016). Comparison of feature selection techniques in machine learning for anatomical brain MRI in dementia. *Neuroinformatics*, *14*(3), 279–296. https://doi.org/10.1007/s12021-015-9292-3

Vanschoren, J., Rijn, J. N. van, Bischl, B., & Torgo, L. (2013). OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, *15*(2), 49–60. https://doi.org/10.1145/2641190.2641198

Venkatesh, B., & Anuradha, J. (2019). A review of feature selection and its methods. *Cybernetics and Information Technologies*, *19*(1), 3–26. https://doi.org/10.2478/cait-2019-0001

Wojtas, M., & Chen, K. (2020). Feature Importance Ranking for Deep Learning. *arXiv:2010.08973 [Cs]*. http://arxiv.org/abs/2010.08973

Yadan, O. (2019). *Hydra - A framework for elegantly configuring complex applications*. https://github.com/facebookresearch/hydra

Yoo, A. B., Jette, M. A., & Grondona, M. (2003). SLURM: Simple Linux Utility for Resource Management. In D. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *Job Scheduling Strategies for Parallel Processing* (pp. 44–60). Springer. https://doi.org/10.1007/10968987_3

Zhao, Z., & Liu, H. (2007). Searching for interacting features. *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, 1156–1161. https://www.semanticscholar.org/paper/Searching-for-Interacting-Features-Zhao-Liu/d2debe138a9b67d838b11d622651383322934aee