Redflag: machine learning safety by design

Matt Hall [1]

1 Equinor, Bergen, Norway

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Monica Bobra

Reviewers:

- @jsheunis
- @kaurao

Submitted: 02 October 2023
Published: unpublished

Summary

Redflag is a Python library that applies "safety by design" to machine learning. It helps researchers and practitioners in this field ensure their models are safe and reliable by alerting them to potential pitfalls. These pitfalls could lead to overconfidence in the model or wildly spurious predictions. *Redflag* offers accessible ways for users to integrate safety checks into their workflows by providing `scikit-learn` transformers, pandas accessors, and standalone functions. These components can easily be incorporated into existing workflows, helping identify issues and enhance the quality and safety of predictive models.

Redflag is distributed under the Apache 2.0 license. The source code is available on GitHub and includes tests and documentation. The package can be installed from the Python package index with `pip install redflag` or using Conda with `conda install -c conda-forge redflag`.

Statement of need

Safety by design means to 'design out' hazardous situations from complex machines or processes before they can do harm. The concept, also known as *prevention through design*, has been applied to civil engineering and industrial design for decades. Recently it has also been applied to software engineering and, more recently still, to machine learning (Gelder et al., 2021). *Redflag* helps machine learning researchers and practitioners design safety into their workflows.

The practice of machine learning features a great many pitfalls that threaten the safe application of the resulting model. These pitfalls vary in the type and seriousness of their symptoms:

1. **Minor issues** resulting in overconfidence in the model (or, equivalently, underperformance of the model compared to expectations), such as having insufficient data, a few spurious data points, or failing to compute feature interactions.
2. **Moderate issues** arising from incorrect assumptions or incorrect application of the tools. Pitfalls include not dealing appropriately with class imbalance, not recoginizing spatial or temporal or other correlation in the data, or overfitting to the training or test data.
3. **Major issues** resulting in egregiously spurious predictions. Causes include feature leakage (using features unavailable in application), using distance-dependent algorithms on unscaled data, or forgetting to scale input features in application.
4. **Critical issues**, especially project design and implementation issues, that result in total failure. For example, asking the wrong question, not writing tests or documentation, not training users of the model, or violating ethical standards.

While some of these pathologies are difficult to check with code (especially those in class 4, above), many of them could in principle be caught automatically by inserting checks into the workflow that trains, evaluates, and implements the predictive model. The goal of *Redflag* is to provide those checks.

In the Python machine learning world, pandas (McKinney, 2010) is the *de facto* tabular data manipulation package, and `scikit-learn` (Pedregosa et al., 2011) is the preeminent prototyping and implementation framework. By integrating with these packages by providing accessors and transformers respectively, *Redflag* aims to be easy to learn and adopt.

Redflag offers three ways for users to insert safety checks into their machine learning workflows:

1. **`scikit-learn` transformers** which fit directly into the pipelines that most data scientists are already using, e.g. `redflag.ImbalanceDetector().fit_transform(X, y)`.
2. **pandas accessors** on Series and DataFrames, which can be called like a method on existing Pandas objects, e.g. `df['target'].redflag.is_imbalanced()`.
3. **Standalone functions** which the user can compose their own checks and tests with, e.g. `redflag.is_imbalanced(y)`.

There are two kinds of `scikit-learn` transformer:

- **Detectors** check every dataset they encounter. For example, `redflag.ClippingDetector` checks for clipped data during both model fitting and during prediction.
- **Comparators** learn some parameter in the model fitting step, then check subsequent data against those parameters. For example, `redflag.DistributionComparator` learns the empirical univariate distributions of the training features, then checks that the features in subsequent datasets are tolerably close to these baselines.

Although the `scikit-learn` components are implemented as transformers, subclassing `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`, they do not transform the data. They only raise warnings (or, optionally, exceptions) when a check fails. *Redflag* does not attempt to fix any problems it encounters.

There are some other packages with similar goals. For example, `great_expectations` provides a full-featured framework with a great deal of capability, especially oriented around cloud services, and a correspondingly large API. Meanwhile, `pandas_dq`, `pandera`, `pandas-profiling` are all oriented around Pandas, Spark or other DataFrame-like structures. Finally, `evidently` provides on a Jupyter interface with lots of plots.

By providing to machine learning practitioners a range of alerts and alarms, each of which can easily be inserted into existing workflows and pipelines, *Redflag* aims to allow anyone to create higher quality, more trustworthy prediction models that are safer by design.

Acknowledgements

Thanks to the Software Underground community for discussion and feedback over the years.

References

Gelder, P. van, Klaassen, P., Taebi, B., Walhout, B., Ommen, R. van, Poel, I. van de, Robaey, Z., Asveld, L., Balkenende, R., Hollmann, F., Kampen, E. J. van, Khakzad, N., Krebbers, R., Lange, J. de, Pieters, W., Terwel, K., Visser, E., Werff, T. van der, & Jung, D. (2021). Safe-by-design in engineering: An overview and comparative analysis of engineering disciplines. *International Journal of Environmental Research and Public Health*, *18*(12), 6329. https://doi.org/10.3390/ijerph18126329

McKinney, Wes. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.