The Machine Learning Benchmark Tools Package

Ryan Turner
Uber AI Labs

Rigorous design of machine learning (ML) challenges is at least as difficult as rigorous design of experiments. Challenges are experiments comparing multiple machine learning algorithms (possibly with adversarial agents). Part of experimental design involves proper analysis with confidence intervals (error bars) and statistical tests.

Surprisingly, error bars and significance levels are rarer in ML than other parts of science, even though ML methodology is often based upon statistics. Challenges and benchmark data sets tend to plateau: Initial big gains are followed by a long period of incremental gains. Once these gains become small, they are often explainable purely by sampling noise.

To enable widespread error analysis with minimal time overhead we present the *Benchmark Tools* Python package.[1] Benchmark Tools is usable as a simple one-liner using a dictionary of sklearn compatible objects [8] and train/test data sets. The routine will train, test, and evaluate the models on multiple loss functions. Individual pieces of this process are usable in a modular way.

Considering multiple loss functions is a design principle of Benchmark Tools. Often challenges and analysis are based upon a single (somewhat arbitrary) loss function, and ML developers often become obsessed with incremental improvements in a single metric. The Benchmark Tools package supports the Bayes' decision rule calculation that converts a predictive distribution into a different *action* for each loss function. The classic example for regression is for MSE one should report the mean of the predictive distribution while for MAE one should report the median [7]. This conversion is done automatically within the package and is essential if one wants to ensure a single model is being benchmarked fairly and consistently across multiple metrics. The classification benchmark can build loss functions with arbitrary loss matrices; this allows, for example, probabilistic predictions to be converted to a "don't know" option and evaluated.

The interface Benchmark Tools was largely motivated by noticing that applied ML projects contain many repeated "boilerplate" code segments that frequently repeat across projects. These typically consist of: splitting data, training models, testing models, evaluating with statistical analysis, and finally proper formatting of tables.

The high level interface of the package has just two phases: `just_benchmark` and `just_format_it`. These work as follows:

```
import benchmark_tools.classification as btc
from benchmark_tools.classification import STD_CLASS_LOSS, STD_BINARY_CURVES
performance_df, performance_curves_dict = \
    btc.just_benchmark(X_train, y_train, X_test, y_test, 2, classifiers,
                       STD_CLASS_LOSS, STD_BINARY_CURVES, ref_method='iid')
```

This benchmarks all the models in the classifiers dictionary `classifiers` on the data `(X_train, y_train, X_test, y_test)` for 2-class classification. It uses the loss function described in the dictionaries `STD_CLASS_LOSS`, and the curves (e.g., ROC, PR) in `STD_BINARY_CURVES`. `ref_method` defines the model that is the reference to compare against (using a paired statistical test giving tighter error bars) for assessing statistically significant performance gains. We provide an iid dummy model (`JustNoise`) that provides constant predictions to serve as a simple baseline. The `classifiers` dictionary is as simple as:

```
classifiers = {'iid': btc.JustNoise(),
               'Nearest Neighbors': KNeighborsClassifier(3),
               'Linear SVM': SVC(kernel='linear', C=0.025, probability=True),
               'RBF SVM': SVC(gamma=2, C=1, probability=True)}
```

The objects need not be sklearn objects but merely support the methods `fit` and `predict_log_proba` as per the sklearn interface. Equivalent routines are available in `benchmark_tools.regression` for regression problems.

Although the most convenient way to benchmark is via the "do-it-all" `just_benchmark` routine, the package also allows modular usage of `just_benchmark`'s three phases. This allows for building a dataframe of: predictive distributions on each test point and model (`get_pred_log_prob`), the losses for each prediction (`loss_table`), and the mean loss for each method along with error bars and p-values (`loss_summary_table`).

[1] This package is found at `https://github.com/rdturnermtl/benchmark_tools`.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Sciprint The `performance_df` is a pandas dataframe with a table summarizing the performance. However for publishable results, one must first format it correctly. The *sciprint* module formats these tables for scientific presentation [2]. The performance dataframe is converted to cleanly formatted tables: correct significant figures, shifting of exponent for compactness, and correct alignment of decimal points, units in headers, etc. Here we use:

```
import benchmark_tools.sciprint as sp
print(sp.just_format_it(performance_df, shift_mod=3, unit_dict={'NLL': 'nats'},
                        non_finite_fmt={sp.NAN_STR: '{--}'}, use_tex=True))
```

Both plain text and LaTeX tables are available via the `use_tex` argument. The above snippet produces the LaTeX table:

	AP	p	AUC	p	AUPRG	p	Brier	p	NLL (nats)	p	sphere	p	zero one	p
Linear SVM	0.952(99)	<0.0001	0.950(77)	<0.0001	0.88705	<0.0001	0.34(24)	<0.0001	0.29(16)	<0.0001	0.31(24)	<0.0001	0.15(12)	0.0006
Nearest Neighbors	0.94(14)	<0.0001	0.969(69)	<0.0001	0.93498	<0.0001	0.18(21)	<0.0001	0.42(70)	0.4241	0.15(18)	<0.0001	0.025(51)	<0.0001
RBF SVM	0.93(18)	<0.0001	0.957(94)	<0.0001	0.92081	<0.0001	0.14(20)	<0.0001	0.18(18)	<0.0001	0.12(17)	<0.0001	0.025(51)	<0.0001
iid	0.53(16)	–	0.5(0)	–	0(0)	–	1.004(22)	–	0.695(11)	–	1.005(27)	–	0.53(17)	–

The sciprint module automatically enforces good practices for presentation of numeric results. Sciprint crops the significant figures on the performance number to match the specified number of error digits. The p-values are limited to specified number of digits. The package even automatically adjusts the exponent of each column to minimize its width for written compactness.

Data splitter The package comes with a *data splitter* module that supports random, ordinal, or temporal splitting across features in pandas dataframes. It also allows for jointly splitting across multiple features to test difficult generalization cases (e.g., test set of random unseen users *and* later in time than training).

Loss functions Benchmark Tools is based upon two types of metrics: *loss functions* and *curve summaries*. From a decision theoretic perspective, loss functions are the more justified metric for evaluation; they are also easier to place confidence intervals on. The loss for model A on iid test data with labels $y_{1:N}$ works as

$$\mathcal{L}_A = \sum_{i=1}^{N} \ell_i = \sum_{i=1}^{N} \ell(y_i, a_i)\,, \quad a_i = \operatorname*{argmin}_a \mathbb{E}_{P_A(y_i|\mathbf{x}_i)}[\ell(y_i, a)]\,, \tag{1}$$

where a_i is the Bayes' optimal action for the loss function ℓ. Benchmark Tools has built-in support for general *loss matrices* for hard classification but also supports the log loss (NLL), Brier loss, and spherical loss as *proper scoring rules* [5] to evaluate a model's soft predictions. The modular system is extendable, allowing for new metrics to be easily added. Non-probabilistic methods are usable by "pipelining" with a *calibrator* [6, 9], some of which are in sklearn.

Curve summaries create a performance curve and then summarize with a single number. We support ROC, precision-recall, and precision-recall-gain [4] curves. Great care is taken to ensure these curves and summaries are unbiased and behave correctly for a random classifier, something that was *not* done in sklearn (e.g., see issue #4577).

Error bars Putting error bars on the loss functions is essentially placing an error bar on the mean of ℓ. Note this places a confidence interval on what the performance would be on a new ($N \to \infty$) test set from the same distribution, and with the same trained model. Given the individual losses $\ell_{1:N}$, we support three methods for confidence intervals: t-test, bootstrap, and Bernstein bound. The t-test is fairly standard, but makes a central limit assumption that the error distribution on the mean is normally distributed. The percentile bootstrap still makes some asymptotic assumptions but weaker than the t-test. For very conservative error bars, we offer the Bernstein bound, which is distribution free and holds for finite sample [1], but requires a *bounded loss*; otherwise, it results in infinite error bars.

Significance tests The p-values are designed to match the error bars, and therefore constructed using either t-test, bootstrap, or Bernstein bound. The confidence interval (U, L) on $\Delta\mathcal{L} := \mathcal{L}_A - \mathcal{L}_R$ and p-value p (on H_0: $\mathcal{L}_A = \mathcal{L}_R$) are designed to be coherent in the sense that: $L = 0$ or $U = 0$ if the confidence interval is computed with coverage $\alpha = 1 - p$. Construction of p-values from the t-test and bootstrap are standard, but Bernstein is more novel.

Error bars on curves The error bars and significance tests for the curves are produced via bootstrap [3]. We place confidence intervals on both the raw curves (for plotting) and their AUC summaries (for tables). We built a vectorized bootstrap that reweights the data points via a multinomial distribution. This avoids re-creating the data sets in memory upon resampling, which is very slow for large data sets. For example, the MATLAB bootstrap routine (`bootstrp`) takes this approach, and is often slower than training the classifier itself for large data sets.

Conclusions We have presented a package with utility to be used in many ML challenges and publications. The use of one-liner performance tables with error bars and significance levels should make its use universal in ML. The package still has much potential for expansion beyond classification and regression to more complex tasks such as structured prediction problems.

Acknowledgments We thank Riashat Islam for setting up continuous integration testing in the Benchmark Tools package, and Jane Hung for proofreading.

References

[1] J.-Y. Audibert, R. Munos, and C. Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009.

[2] T. Cole. Too many digits: The presentation of numerical data. *Archives of Disease in Childhood*, 2015.

[3] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994.

[4] P. Flach and M. Kull. Precision-recall-gain curves: PR analysis done right. In *Advances in Neural Information Processing Systems*, pages 838–846, 2015.

[5] T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[6] M. Kull, T. S. Filho, and P. Flach. Beta calibration: A well-founded and easily implemented improvement on logistic calibration for binary classifiers. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 54, pages 623–631, 2017.

[7] J. Marchini. Lecture notes in foundations of statistical inference, 2013.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 10(3):61–74, 1999.