The Machine Learning Paper (mlpaper) Package

Ryan Turner

Introduction

- Easy benchmarking of machine learning models with sklearn interface with statistical tests built-in
- Error bars and significance tests should be standard in ML
- One liner on sklearn compatible objects
  - Train, test, and evaluate the models on multiple loss functions
- Process is usable in a modular way
- Multiple loss functions is a key design principle
- Bayes' decision rule: predictive distn. $\rightarrow$ *action* for each loss function
- **Example**: MSE $\implies$ mean, MAE $\implies$ median
- Automatic conversion to ensure model fairly compared across metrics

Code `https://github.com/rdturnermtl/mlpaper`

The interface

- Classification uses `mlpaper.classification`
- Regression uses `mlpaper.regression`
- Bayes' decision rule converts predictive distribution to *action* for each loss function
- Objects just support methods `fit` and `predict_log_proba` (sklearn interface)

Modular pieces

- The "do-it-all" `just_benchmark` routine has 3 phases
- `get_pred_log_prob`: predictive distributions on each test point and model
- `loss_table`: the losses for each prediction
- `loss_summary_table`: mean loss for each method + error bars and p-values

Sciprint

- Publishable results: one must first format `performance_df`
- Cleanly format: correct significant figures, shifting of exponent for compactness, and correct alignment of decimal points, units in headers

Data splitter

- Supports random, ordinal, or temporal splitting across features in pandas dataframes
- Jointly splitting across multiple features to test difficult generalization cases

Evaluation framework

- Two metric types: *loss functions* and *curve summaries*

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

- Curve summaries (ROC, PR, PRG) for a single number
- Built-in *proper scoring rules*: log loss (NLL), Brier loss, spherical loss
- General loss matrices, and new metrics are easily added
- Non-probabilistic methods usable by "pipelining" a *calibrator*

Error bars and significance tests

- Place CI on mean loss with $N \rightarrow \infty$ test set from the same distn.
- Three methods for CI: t-test, bootstrap, and Bernstein bound
- The p-values are designed to match the error bars (via the 3 methods)

Error bars on curves

- CI on raw curves (for plotting) and AUC (for tables) via bootstrap
- Vectorized bootstrap: reweight data points via multinomial distribution
- Avoids re-creating the data sets in memory (very slow)

Examples

- Motivation: reduce common "boilerplate" code across ML projects
- Pipeline: split data, training models, testing models, evaluating/statistical analysis, and formatting of tables
- Just two phases: `just_benchmark` and `just_format_it`

Classification

```
import mlpaper.classification as btc
from mlpaper.classification import STD_BINARY_CURVES, STD_CLASS_LOSS
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')
```

The `classifiers` dictionary is as simple as:

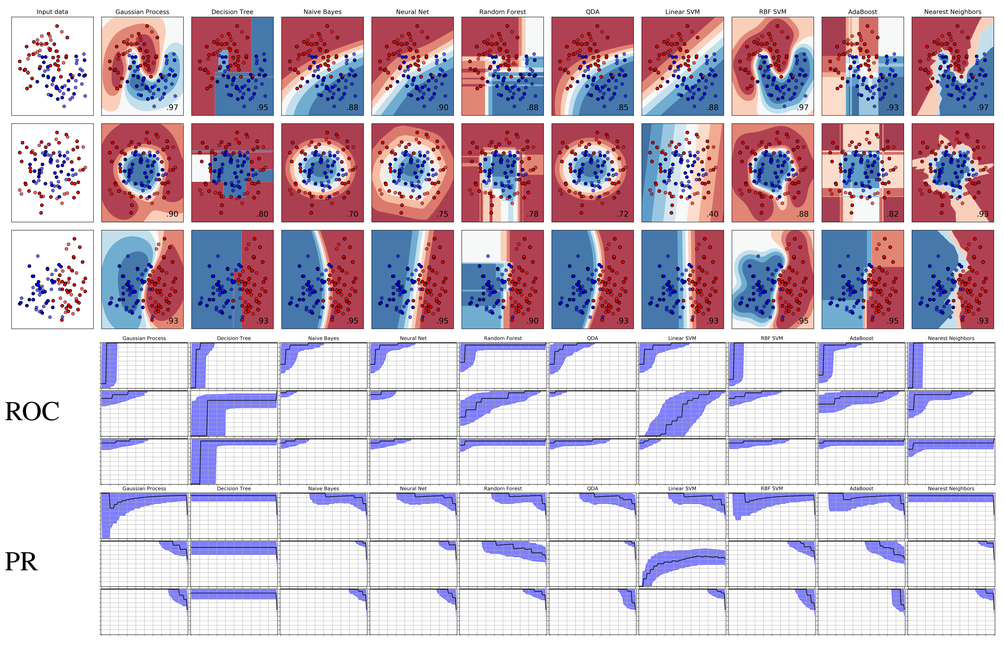
```
classifiers = {'iid': btc.JustNoise(),
               'Nearest Neighbors': KNeighborsClassifier(3),
               'Linear SVM': SVC(kernel='linear', C=0.025, probability=True),
               'RBF SVM': SVC(gamma=2, C=1, probability=True)}
```

- Loss functions in `STD_CLASS_LOSS`, curves (e.g., ROC, PR) in `STD_BINARY_CURVES`
- The reference `ref_method` for paired statistical tests
- Constant prediction dummy model in `JustNoise`

```
import mlpaper.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))
```

	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)	–

Performance curves



Regression

```
import mlpaper.regression as btr
full_tbl = btr.just_benchmark(X_train, y_train, X_test, y_test,
                              regressors, STD_REGR_LOSS, 'iid',
                              pairwise_CI=True)
```

	MAE	p	MSE	p	NLL (nats)	p
BLR	0.96933(30)	0.0979	1.39881(67)	0.0665	1.58842(57)	0.9828
GPR	0.75(13)	0.0009	0.75(28)	<0.0001	1.27(12)	<0.0001
iid	0.96908	–	1.3982	–	1.5884	–