Machine Learning Automation Toolbox (MLaut)

Viktor Kazakov * [1,2] and Franz J. Király † [3,4]

[1] Lab of Innovative Finance and Technology (LIFTech),
Civil, Environmental and Geomatic Engineering Department,
University College London, Gower Street, London WC1E 6BT, United Kingdom
[2] European Bank for Reconstruction and Development
One Exchange Square, London EC2A 2JN, United Kingdom
[3] Department of Statistical Science, University College London,
Gower Street, London WC1E 6BT, United Kingdom
[4] The Alan Turing Institute,
The British Library, Kings Cross, London NW1 2DB, United Kingdom

January 14, 2019

Abstract

In this paper we present MLaut (Machine Learning AUtomation Toolbox) for the python data science ecosystem. MLaut automates large-scale evaluation and benchmarking of machine learning algorithms on a large number of datasets. MLaut provides a high-level workflow interface to machine algorithm algorithms, implements a local back-end to a database of dataset collections, trained algorithms, and experimental results, and provides easy-to-use interfaces to the scikit-learn and keras modelling libraries. Experiments are easy to set up with default settings in a few lines of code, while remaining fully customizable to the level of hyper-parameter tuning, pipeline composition, or deep learning architecture.

As a principal test case for MLaut, we conducted a large-scale supervised classification study in order to benchmark the performance of a number of machine learning algorithms - to our knowledge also the first larger-scale study on standard supervised learning data sets to include deep learning algorithms. While corroborating a number of previous findings in literature, we found (within the limitations of our study) that deep neural networks do not perform well on basic supervised learning, i.e., outside the more specialized, image-, audio-, or text-based tasks.

Contents

*viktor.kazakov.18@ucl.ac.uk
†f.kiraly@ucl.ac.uk

1 Introducing MLaut

MLaut [32] is a modelling and workflow toolbox in python, written with the aim of simplifying large scale benchmarking of machine learning strategies, e.g., validation, evaluation and comparison with respect to predictive/task-specific performance or runtime. Key features are:

(i) automation of the most common workflows for benchmarking modelling strategies on multiple datasets including statistical post-hoc analyses, with user-friendly default settings
(ii) unified interface with support for scikit-learn strategies, keras deep neural network architectures, including easy user extensibility to (partially or completely) custom strategies
(iii) higher-level meta-data interface for strategies, allowing easy specification of scikit-learn pipelines and keras deep network architectures, with user-friendly (sensible) default configurations
(iv) easy setting up and loading of data set collections for local use (e.g., data frames from local memory, UCI repository, openML, Delgado study, PMLB)
(v) back-end agnostic, automated local file system management of datasets, fitted models, predictions, and results, with the ability to easily resume crashed benchmark experiments with long running times

MLaut may be obtained from pyPI via `pip install mlaut`, and is maintained on GitHub at `github.com/alan-turing-institute/mlaut`. A Docker implementation of the package is available on Docker Hub via `docker pull kazakovv/mlaut`.

Note of caution: time series and correlated/associated data samples

MLaut implements benchmarking functionality which provides statistical guarantees under assumption of either independent data samples, independent data sets, or both. This is mirrored in Section 2.3 by the crucial mathematical assumptions of statistical independence (i.i.d. samples), and is further expanded upon in Section 2.4.
In particular, it should be noted that naive application of the validation methodology implemented in MLaut to samples of time series, or other correlated/associated/non-independent data samples (within or between datasets), will in general violate the validation methodologies' assumptions, and may hence result in misleading or flawed conclusions about algorithmic performance.
The BSD license under which MLaut is distributed further explicitly excludes liability for any damages arising from use, non-use, or mis-use of MLaut (e.g., mis-application within, or in evaluation of, a time series based trading strategy).

1.1 State-of-art: modelling toolbox and workflow design

A hierarchy of modelling designs may tentatively be identified in contemporary machine learning and modelling ecosystems, such as the python data science environment and the R language:

Level 1. implementation of specific methodology or a family of machine learning strategies, e.g., the most popular packages for deep learning, Tensorflow [2], MXNet [11], Caffe [31] and CNTK [40].
Level 2. provision of a unified interface for methodology solving the same "task", e.g., supervised learning aka predictive modelling. This is one core feature of the Weka [29], scikit-learn [35] and Shogun [41] projects which both also implement level 1 functionality, and main feature of the caret [47] and mlr [6] packages in R which provides level 2 functionality by external interfacing of level 1 packages.
Level 3. composition and meta-learning interfaces such as tuning and pipeline building, more generally, first-order operations on modelling strategies. Packages implementing level 2 functionality usually (but not always) also implement this, such as the general hyper-parameter tuning and pipeline composition operations found in scikit-learn and mlr or its mlrCPO extension. Keras [12] has abstract level 3 functionality specific to deep learning, Shogun possesses such functionality specific to kernel methods.
Level 4. workflow automation of higher-order tasks performed with level 3 interfaces, e.g., diagnostics, evaluation and comparison of pipeline strategies. Mlr is, to our knowledge, the only existing modelling toolbox with a modular, class-based level 4 design that supports and automates resampling based model evaluation workflows. The Weka GUI and module design also provides some

level 4 functionality.

A different type of level 4 functionality is automated model building, closely linked to but not identical with benchmarking and automated evaluation - similarly to how, mathematically, model selection is not identical with model evaluation. Level 4 interfaces for automated model building also tie into level 3 interfaces, examples of automated model building are implemented in auto-Weka [24], auto-sklearn [20], or extensions to mlrCPO [44].

In the Python data science environment, to our knowledge, there is currently no widely adopted solution with level 4 functionality for evaluation, comparison, and benchmarking workflows. The reasonably well-known skll [1] package provides automation functionality in python for scikit-learn based experiments but follows an unencapsulated scripting design which limits extensibility and usability, especially since it is difficult to use with level 3 functionality from scikit-learn or state-of-art deep learning packages.

Prior studies conducting experiments which are level 4 use cases, i.e., large-scale benchmarking experiments of modelling strategies, exist for supervised classification, such as [19, 45]. Smaller studies, focusing on a couple of estimators trained on a small number of datasets have also been published [28]. However, to the best of our knowledge: none of the authors released a toolbox for carrying out the experiments; code used in these studies cannot be directly applied to conduct other machine learning experiments; and, deep neural networks were not included as part of the benchmark exercises.

At the current state-of-art, hence, there is a distinct need for level 4 functionality in the scikit-learn and keras ecosystems. Instead of re-creating the mlr interface or following a GUI-based philosophy such as Weka, we have decided to create a modular workflow environment which builds on the particular strengths of python as an object oriented programming language, the notebook-style user interaction philosophy of the python data science ecosystem, and the contemporary mathematical-statistical state-of-art with best practice recommendations for conducting formal benchmarking experiments - while attempting to learn from what we believe works well (or not so well) in mlr and Weka.

1.2 Scientific contributions

MLaut is more than a mere implementation of readily existing scientific ideas or methods. We argue that the following contributions, outlined in the manuscript, are scientific contributions closely linked to its creation:

(1) design of a modular “level 4” software interface which supports the predictive model validation/-comparison workflow, a data/model file input/output back-end, and an abstraction of post-hoc evaluation analyses, at the same time.

(2) a comprehensive overview of the state-of-art in statistical strategy evaluation, comparison and comparative hypothesis testing on a collection of data sets. We further close gaps in said literature by formalizing and explicitly stating the kinds of guarantees the different analyses provide, and detailing computations of related confidence intervals.

(3) as a principal test case for MLaut, we conducted a large-scale supervised classification study in order to benchmark the performance of a number of machine learning algorithms, with a key sub-question being whether more complex and/or costly algorithms tend to perform better on real-world datasets. On the representative collection of UCI benchmark datasets, kernel methods and random forests perform best.

(4) as a specific but quite important sub-question we empirically investigated whether common off-shelf deep learning strategies would be worth considering as a default choice on the “average” (non-image, non-text) supervised learning dataset. The answer, somewhat surprising in its clarity, appears to be that they are not - in the sense that alternatives usually perform better. However, on the smaller tabular datasets, the computational cost of off-shelf deep learning architectures is also not as high as one might naively assume. This finding is also subject to a major caveat and future confirmation, as discussed in Section 5.4.3 and Section 5.6.4.

Literature relevant to these contribution will be discussed in the respective sections.

1.3 Overview: usage and functionality

We present a short written demo of core MLaut functionality and user interaction, designed to be convenient in combination with jupyter notebook or scripting command line working style. Introductory jupyter notebooks similar to below may be found as part of MLaut’s documentation [32].

The first step is setting up a database for the dataset collection, which has to happen only once per computer and dataset collection, and which we assume has been already stored in a local MLaut HDF5 database. The first step in the core benchmarking workflow is to define hooks to the database input and output files:

```
input_io = data.open_hdf5(...) #path to input HDF5 file
out_io = data.open_hdf5(...) #path to output HDF5 file
```

After the hooks are created we can proceed to preparing fixed re-sampling splits (training/test) on which all strategies are evaluated. By default MLaut creates a single evaluation split with a uniformly sampled $\frac{2}{3}$ of the data for training and $\frac{1}{3}$ for testing.

```
data.split_datasets(hdf5_in=..., hdf5_out=..., dataset_paths=...)
```

For a simple set-up, a standard set of estimators that come with sensible parameter defaults can be initialized. Advanced commands allow to specify hyper-parameters, tuning strategies, keras deep learning architectures, scikit-learn pipelines, or even fully custom estimators.

```
est = ['RandomForestClassifier','BaggingClassifier']
estimators = instantiate_default_estimators(estimators=est)
>>> estimators
<mlaut.estimators.ensemble_estimators.Random_Forest_Classifier>
<mlaut.estimators.ensemble_estimators.Bagging_Classifier>
```

The user can now proceed to running the experiments. Training, prediction and evaluation are separate; partial results, including fitted models and predictions, are stored and retrieved through database hooks. This allows intermediate analyses, and for the experiment to easily resume in case of a crash or interruption. If this happens, the user would simply need to re-run the code above and the experiment will continue from the last checkpoint, without re-executing prior costly computation.

```
>>> orchest.run(modelling_strategies=estimators)
RandomForestClassifier trained on dataset 1
RandomForestClassifier trained on dataset 2
...
```

The last step in the pipeline is executing post-hoc analyses for the benchmarking experiments. The `AnalyseResults` class allows to specify performance quantifiers to be computed and comparison tests to be carried out, based on the intermediate computation data, e.g., predictions from all the strategies.

```
analyze.prediction_errors(score_accuracy, estimators)
```

The `prediction_errors()` method returns two sets of results: `errors_per_estimator` dictionary which is used subsequently in further statistical tests and `errors_per_dataset _per_estimator_df` which is a dataframe with the loss of each estimator on each dataset that can be examined directly by the user.

We can also use the produced errors in order to perform the statistical tests for method comparison. The code below shows an example of running a t-test.

```
_, t_test_df = analyze.t_test(errors_per_estimator)
>>> t_test_df
Estimator 1        Estimator 2
t_stat  p_val      t_stat  p_val
Estimator 1 ...     ...        ...     ...
Estimator 2 ...     ...        ...     ...
...
```

Data frames or graphs resulting from the analyses can then be exported, e.g., for presentation in a scientific report.

Authors contributions

MLaut is part of VK's PhD thesis project, the original idea being suggested by FK. MLaut and this manuscript were created by VK, under supervision by FK. The design of MLaut is by VK, with suggestions by FK. Sections 1, 2 and 3 were substantially edited by FK before publication, other sections received only minor edits (regarding content). The benchmark study of supervised machine learning strategies was conducted by VK.

Acknowledgments

We thank Bilal Mateen for critical reading of our manuscript, and especially for suggestions of how to improve readability of Section 2.4.

FK acknowledges support by The Alan Turing Institute under EPSRC grant EP/N510129/1.

2 Benchmarking supervised learning strategies on multiple datasets - generative setting

This section introduces the mathematical-statistical setting for the mlaut toolbox - supervised learning on multiple datasets. Once the setting is introduced, we are able to describe the suite of statistical benchmark post-hoc analyses that mlaut implements, in Section 3.

2.1 Informal workflow description

Informally, and non-quantitatively, the workflow implemented by mlaut is as follows: multiple prediction strategies are applied to multiple datasets, where each strategy is fitted to a training set and queried for predictions on a test set. From the test set predictions, performances are computed: performances by dataset, and also overall performances across all datasets, with suitable confidence intervals. For performance across all datasets, quantifiers of comparison ("is method A better than method B overall?") are computed, in the form statistical (frequentist) hypothesis tests, where p-values and effect sizes are reported.

The remainder of this Section 2 introduces the *generative setting*, i.e., statistical-mathematical formalism for the data sets and future situations for which performance guarantees are to be obtained. The reporting and quantification methodology implemented in the mlaut package is described in Section 3 in mathematical language, usage and implementation of these in the mlaut package is described in Section 4.

From a statistical perspective, it should be noted that only a single train/test split is performed for validation. This is partly due to simplicity of implementation, and partly due to the state-of-art's incomplete understanding of how to obtain confidence intervals or variances for re-sampled performance estimates. Cross-validation strategies may be supported in future versions.

A reader may also wonder about whether, even if there is only a single set of folds, should there not be three folds per split (or two nested splits), into tuning-train/tuning-test/test[1]. The answer is: yes, if tuning via re-sample split of the training set is performed. However, in line with current state-of-art understanding and interface design, tuning is considered as part of the prediction strategy. That is, the tuning-train/tuning-test split is strategy-intrinsic. Only the train/test split is extrinsic, and part of the evaluation workflow which mlaut implements; a potential tuning split is encapsulated in the strategy. This corresponds with state-of-art usage and understanding of the wrapper/composition formalism as implemented for example with GridSearchCV in sklearn.

2.2 Notational and mathematical conventions

To avoid confusion between quantities which are random and non-random, we always explicitly say if a quantity is a random variable. Furthermore, instead of declaring the type of a random variable, say X, by writing it out as a measurable function $X : \Omega \to \mathcal{X}$, we say "$X$ is a random variable taking values in $\mathcal{X}$", or abbreviated "X t.v.in $\mathcal{X}$", suppressing mention of the probability space Ω which we assume to be the same for all random variables appearing.

This allows us easily to talk about random variables taking values in certain sets of functions, for example a prediction functional obtained from fitting to a training set. Formally, we will denote the set of functions from a set $\mathcal{X}$ to a set $\mathcal{Y}$ by the type theoretic arrow symbol $\mathcal{X} \to \mathcal{Y}$, where bracketing as in $[\mathcal{X} \to \mathcal{Y}]$ may be added for clarity and disambiguation. E.g., to clarify that we consider a function valued random variable f, we will say for example "let f be a random variable t.v.in $[\mathcal{X} \to \mathcal{Y}]$".

An observant reader familiar with measure theory will notice a potential issue (others may want to skip to the next sub-section): the set $[\mathcal{X} \to \mathcal{Y}]$ is, in general, not endowed with a canonical measure. This is remedied as follows: if we talk about a random variable taking values in $[\mathcal{X} \to \mathcal{Y}]$, it is assumed that the image of the corresponding measurable function $X : \Omega \to [\mathcal{X} \to \mathcal{Y}]$, which may not be all of $[\mathcal{X} \to \mathcal{Y}]$, is a measurable space. This is, for example, the case we substitute training data random variables in a deterministic training functional f, which canonically endows the image of f with the substitution push-forward measure.

[1]What we call the "tuning-test fold" is often, somewhat misleadingly, called a "validation fold". We believe the latter terminology is misleading, since it is actually the final test fold which validates the strategy, not second fold.

2.3 Setting: supervised learning on multiple datasets

We introduce mathematical notation to describe D datasets, and K prediction strategies. As running indices, we will consistently use i for the i^{th} dataset, j for j^{th} (training or test) data point in a given data set, and k for the k^{th} estimator.

The data in the i-th dataset are assumed to be sampled from mutually independent, generative/population random variables $(X^{(i)}, Y^{(i)})$, taking values in feature-label-pairs $\mathcal{X}^{(i)} \times \mathcal{Y}$, where either $\mathcal{Y} = \mathbb{R}$ (regression) or $\mathcal{Y}$ is finite (classification). In particular we assume that the label type is the same in all datasets.

The actual data are i.i.d. samples from the population $(X^{(i)}, Y^{(i)})$, which for notational convenience we assume to be split into a training set $\mathcal{D}_i = \Big((X^{(i)}_{tr,1}, (X^{(i)}_{tr,1}), \dots, ((X^{(i)}_{tr,N_i}, (X^{(i)}_{tr,N_i})\Big)$ and a test set $\mathcal{T}_i = \Big((X^{(i)}_1, Y^{(i)}_1), \dots, (X^{(i)}_{M_i}, Y^{(i)}_{M_i})\Big)$. Note that the training and test set in the i-th dataset are, formally, not "sets" (as in common diction) but ordered tuples of length N_i and M_i. This is for notational convenience which allows easy reference to single data points. By further convention, we will write $Y^{(i)}_\star := \Big(Y^{(i)}_1, \dots, Y^{(i)}_{M_i}\Big)$ for the ordered tuple of test labels.

On each of the datasets, K different prediction strategies are fitted to the training set: these are formalized as random prediction functionals $f_{i,k}$ t.v.in $[\mathcal{X}^{(i)} \to \mathcal{Y}]$, where $i = 1 \dots D$ and $k = 1 \dots K$. We interpret $f_{i,k}$ as the fitted prediction functional obtained from applying the k-th prediction strategy on the i-th dataset where it is fitted to the training set.

Statistically, we make mathematical assumptions to mirror the reasonable intuitive assumptions that there is no active information exchange between different strategies, a copies of a given strategy applied to different data sets: we assume that the random variable $f_{i,k}$ may depend on the training set $\mathcal{D}_i$, but is independent of all other data, i.e., the test set $\mathcal{T}_i$ of the i-th dataset, and training and test sets of all the other datasets. It is further assumed that $f_{i,k}$ is independent of all other fitted functionals $f_{i',k'}$ where $i' \neq i$ and k' is entirely arbitrary. It is also assumed that $f_{i,k}$ is conditionally independent of all $f_{i,k'}$, where $k' \neq k$, given $\mathcal{D}_i$.

We further introduce notation for predictions $\hat{Y}^{(i)}_{j,k} := f_{i,k}(X^{(i)}_j)$, i.e., $\hat{Y}^{(i)}_{j,k}$ is the prediction made by the fitted prediction functional $f_{i,k}$ for the actually observed test label $Y^{(i)}_j$.

For convenience, the same notation is introduced for the generative random variables, i.e., $\hat{Y}^{(i)}_k := f_{i,k}(X^{(i)})$. Similarly, we denote by $\hat{Y}^{(i)}_{\star,k} := (\hat{Y}^{(i)}_{1,k}, \dots, \hat{Y}^{(i)}_{M_i,k})$ the random vectors of length M_i whose entries are predictions for full test sample, made by method k.

2.4 Performance - which performance?

Benchmarking experiments produce performance and comparison quantifiers for the competitor methods. It is important to recognise that these quantifiers are computed to create guarantees for the methods' use on putative *future data*. These guarantees are obtained based on mathematical theorems such as the central limit theorem, applicable under empirically justified assumptions. It is crucial to note that mathematical theorems allow establishing performance guarantees on future data, despite the future data not being available to the experimenter at all. It is also important to note that the future data for which the guarantees are created are different from, and in general not identical to, the test data.

Contrary to occasional belief, performance on the test data in isolation is empirically not useful: without a guarantee it is unrelated to the argument of algorithmic effectivity the experimenter wishes to make.

While a full argument usually *does* involve computing performance on a statistically independent test set, the argumentative reason for this best practice is more subtle than being of interest by itself. It is a consequence of "prediction" performance on the training data not being be a fair proxy for performance on future data. Instead, "prediction" on an unseen (statistically independent) test set is a fair(er) proxy, as it allows for formation of performance guarantees on future data: the test set being unseen allows to leverage the central limit theorems for this purpose.

In benchmark evaluation, it is hence crucial to make precise the relation between the testing setting and the application case on future data - there are two key types of distinctions on the future data application case:

(i) whether in the scenario, a fitted prediction function is to be re-used, or whether it is re-fitted on new data (potentially from a new data source).

(ii) whether in the scenario, the data source is identical with the source of one of the observed datasets, or whether the source is merely a source from the same population as the data sources observed.

Being precise about these distinctions is, in fact, practically crucial: similar to the best practice of not testing on the training set, one needs to be careful about whether a *data source*, or a *fitted strategy* that will occur in the future test case has already been observed in the benchmarking experiment, or not.

We make the above mathematically precise (a reader interested only in an informal explanation may first like skip forward to the subsequent paragraph).

To formalize "re-use", distinction (i) translates to conditioning on the fitted prediction functionals $f_{i,k}$, or not. Conditioning corresponds to prior observation, hence having observed the outcome of the fitting process, therefore "re-using" $f_{i,k}$. Not doing so corresponds to sampling again from the random variable, hence "re-fitting".

To formalize the "data source" distinction, we will assume an i.i.d. process P (taking values in joint distributions over $\mathcal{X}^{(i)} \times \mathcal{Y}$ also selected at random), generating distributions according to which population laws are distributed, i.e., $P_1, \dots, P_D \sim P$ is an i.i.d. sample. The i-th element of this sample, P_i is the (generating) data source for the i-th data set i.e., $(X^{(i)}, Y^{(i)}) \sim P_i$. We stress that P_i takes values in distributions, i.e., P_i is a distribution which is itself random[2] and from which data are generated. In this mathematical setting, the distinction (ii) then states whether the guarantee applies for data sampled from $(X^{(i)}, Y^{(i)})$ with a specific i, or instead data sampled from $(X^*, Y^*) \sim P$. The former is "data from the already observed i-th source, the latter is "data from a source similar to, but not identical to, the observed source". If the latter is the case, the same generative principle is applied to yield a prediction functional $f_{*,k}$, drawn i.i.d. from a hypothetical generating process which yielded the $f_{i,k}$ on the i-th dataset. We remain notationally consistent by defining $\hat{Y}^*_k := f_{*,k}(X^*)$.

For **intuitive** clarity, let us consider an **example**: three supervised classification methods, a random forest, logistic regression, and the baseline "predicting the majority class" are benchmarked on 50 datasets, from 50 hospitals, one dataset corresponding to observations in exactly one hospital. Every dataset is a sample of patients (data frame rows) for which as variables (data frame rows) the outcome (= prediction target and data frame column) therapy success yes/no for a certain disease is recorded, plus a variety of demographic and clinical variables (data frame columns) - where what is recorded differs by hospital.

A benchmarking experiment may be asked to produce a performance quantifier for one of the following three distinct key future data scenarios:

(a) re-using the trained classifiers (e.g., random forest), trained on the training data of hospital 42, to make predictions on future data observed in hospital 42.

(b) (re-)fitting a given classifier (e.g., random forest) to new data from hospital 42, to make predictions on further future data observed in hospital 42.

(c) obtaining future data from a new hospital 51, fitting the classifiers to that data, and using the so fitted classifiers to make predictions on further future data observed from hospital 51.

It is crucial to note that both performances and guarantees may (and in general will) differ between these three scenarios. In hospital 42, a random forests may outperform logistic regression and the baseline, while in hospital 43 nothing outperforms the baseline. The behaviour and ranking of strategies may also be different, depending on whether classifiers are re-used, or re-fitted. This may happen in the same hospital, or when done in an average unseen hospital. Furthermore, the same qualitative differences as for observed performances may hold for the precision of the statistical guarantees obtained from performances in a benchmarking experiment: the sample size of patients in a given hospital may be large enough or too small to observe a significant difference of performances in a given hospital, while the sample size of hospitals is the key determinant of how reliable statistical guarantees about performances and performance differences for unseen hospitals are.

In the subsequent, we introduce abbreviating **terminology** for denoting the distinctions above: for (i), we will talk about *re-used* (after training once) and *re-trained* (on new data) prediction algorithm. For (ii), we will talk about *seen* and *unseen* data sources. Further, we will refer to the three future data scenarios abbreviatingly by the letters (a), (b), and (c). By terminology, in these scenarios the algorithm is: (a) re-used on seen sources, (b) re-trained on seen sources, and (c) re-trained on an unseen source (similar to but not identical to seen sources).

[2] Thus, the symbol $\sim$ is used here in its common "distribution" and not "distribution of random variable" meaning which are usually confounded by abuse of notation

It should be noted that it is impossible to re-use an algorithm on an unseen source, by definition of the word "unseen", hence the hypothetical fourth combination of the two dichotomies re-used/re-trained and unseen/seen is logically impossible.

2.5 Performance quantification

Performance of the prediction strategy is measured by a variety of quantifiers which compare predictions for the test set with actual observations from the test set, the "ground truth". Three types of quantifiers are common:

(i) Average loss based performance quantifiers, obtained from a comparison of one method's predictions and ground truth observations one-by-one. An example is the mean squared error on the test set, which is the average squared loss.

(ii) Aggregate performance quantifiers, obtained from a comparison of all of a given method's predictions with all of the ground truth observations. Examples are sensitivity or specifity.

(iii) Ranking based performance quantifiers, obtained from relative performance ranks of multiple methods, from a ranked comparison against each other. These are usually leveraged for comparative hypothesis tests, and may or may not involve computation of ranks based on average or aggregate performances as in (i) and (ii). Examples are the Friedman rank test to compare multiple strategies.

The three kinds of performance quantifiers are discussed in more detail below.

2.5.1 Average based performance quantification

For this, the most widely used method is a loss (or score) function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, which compares a single prediction (by convention the first argument) with a single observation (by convention the second argument).

Common examples for such loss/quantifier functions are listed below in Table 1.

task	name	loss/quantifier function
classification (det.)	MMCE	$L(\widehat{y}, y) = 1 - \mathbb{1}[y = \widehat{y}]$
regression	squared loss	$L(\widehat{y}, y) = (y - \widehat{y})^2$
	absolute loss	$L(\widehat{y}, y) = \lvert y - \widehat{y}\rvert$
	Q-loss	$L(\widehat{y}, y) = \alpha \cdot m(\widehat{y}, y) + (1-\alpha) \cdot m(y, \widehat{y})$ where $m(x, z) = \min(x - z, 0)$

Table 1: List of some popular loss functions to measure prediction goodness (2nd column) used in the most frequent supervised prediction scenarios (1st column). Above, y and $\widehat{y}$ are elements of $\mathcal{Y}$. For classification, $\mathcal{Y}$ is discrete; for regression, $\mathcal{Y} = \mathbb{R}$. The symbol $\mathbb{1}[A]$ evaluates to 1 if the boolean expression A is true, otherwise to 0.

In direct alignment with the different future data scenarios discussed in Section 2.4, the distributions of three generative random variables are of interest:

(a) The conditional random variable $L(f_{i,k}(X^{(i)}), Y^{(i)})|f_{i,k} = L(\hat{Y}_k^{(i)}, Y^{(i)})|f_{i,k}$, the loss when predicting on future data from the i-th data source, when re-using the already trained prediction functional $f_{i,k}$. Note that formally, through conditioning $f_{i,k}$ is implicitly considered constant (not random), therefore reflects re-use of an already trained functional.

(b) The random variable $L(f_{i,k}(X^{(i)}), Y^{(i)}) = L(\hat{Y}_k^{(i)}, Y^{(i)})$, the loss when re-training method k on training data from the i-th data source, and predicting labels on future data from the i-th data source. Without conditioning, no re-use occurs, and this random variable reflects repeating the whole random experiment including re-training of $f_{i,k}$.

(c) The random variable $L(f_{*,k}(X^*), Y^*) = L(\hat{Y}_k^*, Y^*)$, the loss when training method k on a completely new data source, and predicting labels on future data from the same source as that dataset.

The distributions of the above random variables are generative, hence unknown. In practice, the validation workflow estimates summary statistics of these. Of particular interest in the mlaut workflow are related expectations, i.e., (arithmetic) population average errors. We list them below, suppressing notational dependency on L for ease of notation:

(a.1) $\eta_{i,k} := \mathbb{E}[L(\hat{Y}_k^{(i)}, Y^{(i)})|f_{i,k}]$, the (training set) *conditional* expected generalization error of (a re-used) $f_{i,k}$, on data source k.

(a.2) $\overline{\eta}_k := \frac{1}{D}\sum_{i=1}^{D} \mathbb{E}[L(\hat{Y}_k^{(i)}, Y^{(i)})]|(f_{i,k})_{i=1\dots D}$, the *conditional* expected generalization error of the (re-used) k-th strategy, averaged over all *seen* data sources.

(b) $\varepsilon_{i,k} := \mathbb{E}[L(\hat{Y}_k^{(i)}, Y^{(i)})]$, the *unconditional* expected generalization error of (a re-trained) $f_{i,k}$, on data source k.

(c) $\varepsilon_k^* := \mathbb{E}[L(\hat{Y}_k^*, Y^*)]$, the expected generalization error on a typical (unseen) data source.

It should be noted that $\eta_{i,k}$ and $\overline{\eta}_k$ are random quantities, but conditionally constant once the respective $f_{i,k}$ are known (e.g., once $f_{i,k}$ has been trained). It further holds that $\eta_{i,k} = \mathbb{E}[\varepsilon_{i,k}]$.

The mlaut toolbox currently implements estimators for only two of the above three future data situations - namely, only for situations (a: re-used, seen) and (c: re-trained, unseen), i.e., estimators for all quantities with the exception of $\varepsilon_{i,k}$. The reason for this is that for situation (b: re-trained, seen), at the current state of literature it appears unclear how to obtain good estimates, that is, with provably favourable statistical properties independent of the data distribution or the algorithmic strategy. For situations (a) and (c), classical statistical theory may be leveraged, e.g., mean estimation and frequentist hypothesis testing.

It should also be noted that $\varepsilon_{i,k}$ is a *single dataset* performance quantifier rather than a *benchmark* performance quantifier, and therefore outside the scope of mlaut's core use case. While $\eta_{i,k}$ is also a single dataset quantifier, it is easy to estimate en passant while estimating the benchmark quantifier $\overline{\eta}_k$, hence included in discussion as well as in mlaut's functionality.

2.5.2 Aggregate based performance quantification

A somewhat less frequently used alternative are aggregate loss/score functions $L : (\mathcal{Y}\times\mathcal{Y})^+ \to \mathbb{R}$, which compare a tuple of predictions with a tuple of observations in a way that is not expressible as a mean loss such as in Section 2.5.1. Here, by slight abuse of notation, $(\mathcal{Y}\times\mathcal{Y})^+$ denotes tuples of $\mathcal{Y}$-pairs, of fixed length. The use of the symbol L is discordant with the previous section and assumes a case distinction on whether an average or an aggregate is used.

The most common uses of aggregate performance quantifiers are found in deterministic binary classification, as entries of the classification contingency table. These, and further common examples are listed below in Table 2.

task	name	loss/quantifier function
classification (det., binary)	sensitivity, recall	$L(\widehat{y}, y) = \langle y, \widehat{y}\rangle / \|y\|_1$
	specificity	$L(\widehat{y}, y) = \langle \mathbb{1}-y, \mathbb{1}-\widehat{y}\rangle / \|\mathbb{1}-y\|_1$
	precision, PPV	$L(\widehat{y}, y) = \langle y, \widehat{y}\rangle / \|\widehat{y}\|_1$
	F1 score	$L(\widehat{y}, y) = 2\langle y, \widehat{y}\rangle / \langle \mathbb{1}, \widehat{y}+y\rangle$
regression	root mean squared error	$L(y, \widehat{y}) = \|y-\widehat{y}\|_2$

Table 2: List of some popular aggregate performance measues of prediction goodness (2nd column) used in the most frequent supervised prediction scenarios (1st column). Overall, the test sample size is assumed to be M. Hence above, both y and $\widehat{y}$ are elements of $\mathcal{Y}^M$. For binary classification, $\mathcal{Y} = \{0, 1\}$ without loss of generality, 1 being the "positive" class; for regression, $\mathcal{Y} = \mathbb{R}$. By convention, y denotes the true value, and $\widehat{y}$ denotes the prediction. We use vector notation for brevity: $\langle ., .\rangle$ denotes the vector/scalar/inner product, $\|.\|_p$ denotes the p-norm, and $\mathbb{1}$ the M-vector with entries all equal to the number 1.

As before, for the different future data scenarios in Section 2.4, the distributions of three types of generative random variables are of interest. The main complication is that aggregate performance metrics take multiple test points and predictions as input, hence to specify a population performance one must specify a test set size. In what follows, we will fix a specific test set size, M_i, for the i-th dataset. Recall the notation $Y_\star^{(i)}$ for the full vector of test labels on data set i. In analogy, we abbreviatingly denote by $\hat{Y}_{\star,k}^{(i)}$ random vectors of length M_i whose entries are predictions for full test sample, made by method k, i.e., having as the j-th entry to predictions $\hat{Y}_{j,k}^{(i)}$, as introduced in Section 2.3. Similarly, we denote by

$Y^*_\star$ and $\hat{Y}^*_\star$ vectors whose entries, are i.i.d. from the data generating distribution of the new data source, and both of length M^*, which is by assumption the sampling distribution of the M_i.

The population performance quantities of interest can be formulated in terms of the above:

(a.1) $\eta_{i,k} := \mathbb{E}[L(\hat{Y}^{(i)}_{\star,k}, Y^{(i)}_\star)|f_{i,k}]$, the (training set) *conditional* expected generalization error of (a re-used) $f_{i,k}$, on data source k.

(a.2) $\overline{\eta}_k := \frac{1}{D}\sum_{i=1}^D \mathbb{E}[L(\hat{Y}^{(i)}_{\star,k}, Y^{(i)}_\star)]|(f_{i,k})_{i=1\dots D}$, the *conditional* expected generalization error of the (re-used) k-th strategy, averaged over all *seen* data sources.

(b) $\varepsilon_{i,k} := \mathbb{E}[L(\hat{Y}^{(i)}_{\star,k}, Y^{(i)}_\star)]$, the *unconditional* expected generalization error of (a re-trained) $f_{i,k}$, on data source k.

(c) $\varepsilon^*_k := \mathbb{E}[L(\hat{Y}^*_{\star,k}, Y^*_\star)]$, the expected generalization error on a typical (unseen) data source.

As before, the future data situations are (a: re-used algorithm, seen sources), (b: re-trained, seen), and (c: re-trained, unseen). In the general setting, the expectations in (a) and (b) may or may not converge to sensible values as M_i approaches infinity, depending on properties of L. General methods of estimating these depend on availability of test data, which due to the complexities arising and the currently limited state-of-art are outside the scope of mlaut. This unfortunately leaves benchmarking quantity $\overline{\eta}_k$ outside the scope for aggregate performance quantifiers. For (c), classical estimation theory of the mean applies.

2.5.3 Ranking based performance quantification

Ranking based approaches consider, on each dataset, a performance ranking of the competitor strategies with respect to a chosen raw performance statistic, e.g., an average or an aggregate performance such as RMSE or F1-score. Performance assessment is then based on the rankings - in the case of ranking, this is most often a comparison, usually in the form of a frequentist hypothesis test. Due to the dependence of the ranking on a raw performance statistic, it should always be understood that ranking based comparisons are with respect to the chosen raw performance statistic, and may yield different results for different raw performance statistics.

Mathematically, we introduce the population performances in question. Denote $L^{(i)}_k := L(\hat{Y}^{(i)}_k, Y^{(i)}_j)$ in the case the raw statistic being an average, and denote $L^{(i)}_k := L(\hat{Y}^{(i)}_{\star,k}, Y^{(i)})_\star)$ in case it is an aggregate (on the RHS using notation of the respective previous Sections 2.5.1 and 2.5.2). The distribution of $L^{(i)}_k$ models generalization performance of the k-th strategy on the i-th dataset.

We further define rankings $R^{(i)}_k$ as the order rank of $L^{(i)}_k$ within the tuple $(L^{(i)}_1, \dots, L^{(i)}_K)$, i.e., the ranking of the performance $L^{(i)}_k$ within all K strategies' performances on the i-th dataset.

Of common interest in performance quantification and benchmark comparison are the average ranks, i.e., ranks of a strategy averaged over datasets. The population quantity of interest is the expected average rank on a typical dataset, i.e., $r_k := \mathbb{E}[R^{(*)}_k]$, where $R^{(*)}_k$ is the population variable corresponding to sample variables $R^{(i)}_k$. It should be noted that the average rank depends not only on what the k-th strategy is or does, but also on the presence of the other $(K-1)$ strategies in the benchmarking study - hence it is not an absolute performance quantifier for a single method, but a relative quantifier, to be seen in the context of the competitor field.

Common benchmarking methodology of the ranking kind quantifies relative performance on the data sets observed in the sense of future data scenario (b) or (c), where the performance is considered including (re-)fitting of the strategies.

3 Benchmarking supervised learning strategies on multiple datasets - methods

We now describe the suite of performance and comparison quantification methods implemented in the mlaut package. It consists largely of state-of-art of model comparison strategies for the multiple datasets situation, supplemented by our own constructions based on standard statistical estimation theory where appropriate. References and prior work will be discussed in the respective sub-sections. mlaut supports the following types of benchmark quantification methodology and post-hoc analyses:

(i) loss-based performance quantifiers, such as mean squared error and mean absolute error, including confidence intervals.

(ii) aggregate performance quantifiers, such as contingency table quantities (sensitivity, specifity) in classification, including confidence intervals.

(iii) rank based performance quantifiers, such as average performance rank.

(iv) comparative hypothesis tests, for relative performance of methods against each other.

The exposition uses notation and terminology previously introduced in Section 2. Different kinds of quantifiers (loss and/or rank based), and different kinds of future performance guarantees (trained vs re-fitted prediction functional; seen vs unseen sources), as discussed in Section 2.4, may apply across all types of benchmarking analyses.

Which of these is the case, especially under which future data scenario the guarantee given is supposed to hold, will be said explicitly for each, and should be taken into account by any use of the respective quantities in scientific argumentation.

Practically, our recommendation is to consider which of the future data scenarios (a), (b), (c) a guarantee is sought for, and whether evidencing differences in rank, or differences in absolute performances, are of interest.

3.1 Average based performance quantifiers and confidence intervals

For average based performance quantifiers, performances and their confidence intervals are estimated from the sample of loss/score evaluates. We will denote the elements in this sample by $L_{j,k}^{(i)} := L(\widehat{Y}_k^{(i)}, Y_j^{(i)})$ (for notation on RHS see Section 2.5.1). Note that, differently from the population quantities, there are three (not two) indices: k for the strategy, i for the dataset, and j for which test set point we are considering.

estimate	estimates	f.d.s.	standard error estimate	CLT in
$\widehat{\eta}_{i,k} := \frac{1}{M_i}\sum_{j=1}^{M_i} L_{j,k}^{(i)}$	$\eta_{i,k}$	(a)	$\sqrt{\frac{\widehat{v}_{i,k}}{M_i}}$, where $\widehat{v}_{i,k} := \frac{\sum_{j=1}^{M_i}(L_{j,k}^{(i)} - \widehat{\eta}_{i,k})^2}{M_i - 1}$	M_i
$\widehat{\eta}_k := \frac{1}{D}\sum_{i=1}^{D} \widehat{\eta}_{i,k}$	$\overline{\eta}_k$	(a)	$\frac{1}{D}\sqrt{\sum_{i=1}^{D}\frac{\widehat{v}_{i,k}}{M_i}}$	$D, M_1, \dots, M_D$
$\widehat{\varepsilon}_k^* := \widehat{\eta}_k$	ε_k^*	(c)	$\sqrt{\frac{\widehat{w}_k}{D}}$, where $\widehat{w}_k := \frac{\sum_{i=1}^{D}(\widehat{\eta}_{i,k} - \widehat{\varepsilon}_k^*)^2}{D-1}$	D

Table 3: Table of basic estimates of expected loss, with confidence intervals. First column = definition of the estimate. Second column = the quantity which is estimated by the estimate. Third column = which future data scenario (f.d.s.) estimate and confidence intervals give a guarantee for. Fourth column = standard error estimate (normal approximation) for the estimate in the first column, e.g., to construct frequentist confidence intervals. Fifth column = quantities governing central limit theorem (CLT) asymptotics for the confidence intervals.

Table 3 presents a number of expected loss estimates with proposed standard error estimates. As all estimates are mean estimates of independent (or conditionally independent) quantities, normal approximated, two-sided confidence intervals may be obtained for any of the quantities in the standard way, i.e., at α confidence as the interval

$$[\widehat{\theta} + \Phi^{-1}(\alpha/2)\widehat{SE}, \widehat{\theta} - \Phi^{-1}(\alpha/2)\widehat{SE}]$$

where $\widehat{\theta}$ is the respective (mean) estimate and $\widehat{SE}$ is the corresponding standard error estimate.

Note that different estimates and confidence intervals arise through the different future data scenarios that the guarantee is meant to cover - see Sections 2.5.1 and 2.4 for a detailed explanation how precisely the future data scenarios differ in terms of re-fitting/re-using the prediction functional, and obtaining performance guarantees for predictive use on an unseen/seen data source. In particular, choosing a different future data scenario may affect the confidence intervals even though the midpoint estimate is the same: the midpoint estimates $\widehat{\varepsilon}^*_k$ and $\widehat{\varepsilon}_k$ coincide, but the confidence intervals for future data scenario (c), i.e., new data source and the strategy is re-fitted, are usually wider than the confidence intervals for the future data scenario (a), i.e., already seen data source and no re-fitting of the strategy.

Technically, all expected loss estimates proposed in Table 3 are (conditional) mean estimates. The confidence intervals for $\widehat{\eta}_{i,k}$ and $\widehat{\varepsilon}^*_k$ are obtained as standard confidence intervals for a (conditionally) independent sample mean: $\widehat{\varepsilon}^*_k$ is considered to be the mean of the independent samples $\widehat{\eta}_{i,k}$ (varying over i). $\eta_{i,k}$ is considered to be the mean of the conditionally independent samples $L^{(i)}_{j,k}$ (varying over j, and conditioned on $f_{i,k}$). Confidence intervals for $\widehat{\eta}_k$ are obtained averaging the estimated variances of independent summands $\widehat{\eta}_{i,k}$, which corresponds to the plug-in estimate obtained from the equality $\mathrm{Var}(\widehat{\eta}_k) = \frac{1}{D^2}\sum_{i=1}^D \mathrm{Var}(\widehat{\eta}_{i,k})$ (all variances conditional on the $f_{i,k}$).

3.2 Aggregate based performance quantifiers and confidence intervals

For aggregate based performance quantifiers, performances and their confidence intervals are estimated from the sample of loss/score evaluates. We will denote the elements in this sample by $L^{(i)}_k := L(\hat{Y}^{(i)}_{\star,k}, Y^{(i)}_\star)$ (for notation on RHS see Section 2.5.2). We note that unlike in the case of average based evaluation, there is no running index for the test set data point, only indices i for the data set and k for the prediction strategy.

estimate	estimates	f.d.s.	standard error estimate	CLT in
$\widehat{\varepsilon}^*_k := \frac{1}{D}\sum_{i=1}^D L^{(i)}_k$	ε^*_k	(c)	$\sqrt{\frac{\widehat{w}_k}{D}}$, where $\widehat{w}_k := \frac{\sum_{i=1}^D (L^{(i)}_k - \widehat{\varepsilon}^*_k)^2}{D-1}$	D

Table 4: Table of basic estimates of expected loss, with confidence intervals. First column = definition of the estimate. Second column = the quantity which is estimated by the estimate. Third column = which future data scenario (f.d.s.) estimate and confidence intervals give a guarantee for. Fourth column = standard error estimate (normal approximation) for the estimate in the first column, e.g., to construct frequentist confidence intervals. Fifth column = quantities governing central limit theorem (CLT) asymptotics for the confidence intervals.

Table 4 presents one estimate of expected loss estimates with proposed standard error estimate, for future data situation (c), i.e., generalization of performance to a new dataset. Even though there is only a single estimate, we present it in a table for concordance with Table 3. An confidence interval at α confidence is obtained as

$$[\widehat{\varepsilon}^*_k + \Phi^{-1}(\alpha/2)\widehat{w}_k, \widehat{\varepsilon}^*_k - \Phi^{-1}(\alpha/2)\widehat{w}_k].$$

The mean and variance estimates are obtained from standard theory of mean estimation, by the same principle as $\widehat{\varepsilon}^*_k$ for average based estimates. Estimates for situations (a) may be naively constructed from multiple test sets of the same size, or obtained from further assumptions on L via re-sampling, though we abstain from developing such an estimate as it does not seem to be common - or available - at the state-of-art.

3.3 Rank based performance quantifiers

mlaut has functionality to compute rankings based on any average or aggregate performance statistic, denoted L below. I.e., for any choice of L, the following may be computed.

As in Section 2.5.3, define $L^{(i)}_k := L(\hat{Y}^{(i)}_k, Y^{(i)}_j)$ in the case the raw statistic being an average, and $L^{(i)}_k := L(\hat{Y}^{(i)}_{\star,k}, Y^{(i)}_\star)$ in case it is an aggregate. Denote by $R^{(i)}_k$ the order rank of $L^{(i)}_k$ within the tuple $(L^{(i)}_1, \dots, L^{(i)}_K)$.

Table 5 presents an average rank estimates and an average rank difference estimate, for future data situation (c), i.e., generalization of performance to a new dataset.
The average rank estimate and its standard error is based on the central limit theorem in the number of data sets. The average rank difference estimate is Neményi's critical difference as referred to in [16] which is used in visualizations.

estimate	estimates	f.d.s.	standard error estimate	CLT in
$\widehat{r}_k = \frac{1}{D}\sum_{i=1}^D R_k^{(i)}$	r_k	(c)	$\sqrt{\frac{\widehat{\nu}_k}{D}}$, where $\widehat{\nu}_k := \frac{\sum_{i=1}^D (L_k^{(i)} - \widehat{r}_k)^2}{D-1}$	D
$z_{k,k'} = \widehat{r}_k - \widehat{r}_{k'}$	$r_k - r_{k'}$	(c)	$\frac{z_{k,k'}\cdot\sqrt{6D}}{\sqrt{K(K+1)}}$	K, D

Table 5: Table of basic estimates of average rank, with confidence intervals. First column = definition of the estimate. Second column = the quantity which is estimated by the estimate. Third column = which future data scenario (f.d.s.) estimate and confidence intervals give a guarantee for. Fourth column = standard error estimate (normal approximation) for the estimate in the first column, e.g., to construct frequentist confidence intervals. Fifth column = quantities governing central limit theorem (CLT) asymptotics for the confidence intervals.

3.4 Statistical tests for method comparison

While the methods in previous sections compute performances with confidence bands, they do not by themselves allow to compare methods in the sense of ruling out that differences are due to randomness (with the usual statistical caveat that this can never be ruled out entirely, but the plausibility can be quantified).

mlaut implements significance tests for two classes of comparisons: absolute performance differences, and average rank differences, in future data scenario (c), i.e., with a guarantee for the case where the strategy is re-fitted to a new data source.

mlaut's selection follows closely, and our exposition below follows loosely, the work of [16]. While the latter is mainly concerned with classifier comparison, there is no restriction-in-principle to leverage the same testing procedures for quantitative comparison with respect to arbitrary (average or aggregate) raw performance quantifiers.

3.4.1 Performance difference quantification

The first class of tests we consider quantifies, for a choice of aggregate or average loss L, the significance of average differences of expected generalization performances, between two strategies k and k'. The meanings of "average" and "significant" may differ, and so does the corresponding effect size - these are made precise below.

All the tests we describe are based on the paired differences of performances, where the pairing considered is the pairing through datasets. That is, on dataset i, there are performances of strategy k and k' which are considered as a pair of performances. For the paired differences, we introduce abbreviating notation $\Delta_{k,k'}^{(i)} := \widehat{\eta}_{i,k} - \widehat{\eta}_{i,k'}$ if the performance is an average loss/score, and $\Delta_{k,k'}^{(i)} := L_k^{(i)} - L_{k'}^{(i)}$ if the loss is an aggregate loss/score. Non-parametric tests below will also consider the ranks of the paired differences, we will write $\Lambda_{k,k'}^{(i)}$ for the rank of $\Delta_{k,k'}^{(i)}$ within the sample $(\Delta_{k,k'}^{(1)}, \dots, \Delta_{k,k'}^{(D)})$, i.e., taking values between 1 and D.
We denote by $\Delta_{k,k'}^{(*)}$ and $\Lambda_{k,k'}^{(*)}$ the respective population versions, i.e., the performance difference on a random future dataset, as in scenario (c).

name	tests null	e.s.(raw)	e.s.(norm)	stat.
paired t-test	$\mathbb{E}[\Delta_{k,k'}^{(*)}] \stackrel{?}{=} 0$	$\overline{\Delta}_{k,k'} := \frac{1}{D}\sum_{i=1}^D \Delta_{k,k'}^{(i)}$	$d_{k,k'} := \frac{\overline{\Delta}_{k,k'}}{\sqrt{\widehat{v}_{k,k'}}}$, where $\widehat{v}_{k,k'} = \frac{\sum_{i=1}^D (\Delta_{k,k'}^{(i)} - \overline{\Delta}_{k,k'})^2}{D-1}$	$t_{k,k'} := \sqrt{D}\cdot d_{k,k'}$
Wilcoxon signed-rank t.	$\operatorname{med}[\Lambda_{k,k'}^{(*)}] \stackrel{?}{=} 0$	$w_{k,k'} := \frac{1}{D}\sum_{i=1}^D \Lambda_{k,k'}^{(i)} \operatorname{sgn}(\Delta_{k,k'}^{(i)})$	$\rho_{k,k'} := \frac{2w_{k,k'}}{D+1}$	$W_{k,k'} := D\cdot w_{k,k'}$

Table 6: Table of pairwise comparison tests for benchmark comparison. name = name of the testing procedure. tests null = the null hypothesis that is tested by the testing procedure. e.s.(raw) = the corresponding effect size, in raw units. e.s.(norm) = the corresponding effect size, normalized. stat. = the test statistic which is used in computation of significance. Symbols are defined as in the previous sections.

Table 6 lists a number of common testing procedures. The significances may be seen as guarantees for future data situation (c). The normalized effect size for the paired t-test comparing the performance of strategies k and k', the quantity $d_{k,k'}$ in Table 6, is called Cohen's d(-statistic) for paired samples (to avoid confusion in comparison with literature, it should be noted that Cohen's d-statistic also exists for unpaired versions of the t-test which we do not consider here in the context of performance comparison). The normalized effect size for the Wilcoxon signed-rank test, the quantity $\rho_{k,k'}$, is called biserial rank correlation, or rank-biserial correlation.

It should also be noted that the Wilcoxon signed-rank test, while making use of rank differences, is not a pairwise comparison of strategies' performance ranks - this is a common misunderstanding. While "ranks" appear in both concepts, the ranks in the Wilcoxon signed-rank tests are the ranks of the performance differences, pooled *across* data sets, while in a rank based performance quantifier, the ranking of different methods' performances (not differences) *within* a data sets (not across data sets) is considered.

The above tests are implemented for one-sided and two-sided alternatives. See [37], [16], or [46] for details.

Portmanteau tests for the above may be based on parametric ANOVA, though [16] recommends avoiding these due to the empirical asymmetry and non-normality of loss distributions. Hence for multiple comparisons, mlaut implements Bonferroni and Bonferroni-Holm significance correction based post-hoc testing.

In order to compare the performance of the prediction functions f one needs to perform statistical tests on the output produced by $L(f(X^*), Y^*)$. Below we enumerate the statistical tests that can be employed to assess the results produced by the loss functions L as described in 2.5.1.

3.4.2 Performance rank difference quantification

Performance rank based testing uses the observed performance ranks $R_k^{(i)}$ of the k-th strategy, on the i-th data set. These are defined as above in Section 3.3, of which we keep notation, including notation for the average rank estimate $\widehat{r}_k = \frac{1}{D}\sum_{i=1}^D R_k^{(i)}$. We further introduce abbreviating notation for rank differences, $S_{k,k'}^{(i)} := R_k^{(i)} - R_{k'}^{(i)}$.

name	tests null	e.s.(raw)	e.s.(norm)	stat.
sign test	$\mathbb{E}[\mathrm{sgn}(R_k^{(*)} - R_{k'}^{(*)})] \overset{?}{=} 0$	$S_{k,k'} := \sum_{i=1}^D S_{k,k'}^{(i)}$	$z_{k,k'} := \frac{\sqrt{D}\cdot S_{k,k'}}{\sqrt{D^2 - S_{k,k'}^2}}$	$p_{k,k'} := \frac{D+S_{k,k'}}{2D}$
Friedman test	$r_k - r_{k'} \overset{?}{=} 0$ (for some k, k')	$Q := \frac{12D}{K(K+1)}\sum_{k=1}^K (\widehat{r}_k - \frac{K+1}{2})^2$		$F := \frac{(D-1)Q}{D(K-1)-Q}$

Table 7: Table of pairwise comparison tests for benchmark comparison. name = name of the testing procedure. tests null = the null hypothesis that is tested by the testing procedure. e.s.(raw) = the corresponding effect size, in raw units. e.s.(norm) = the corresponding effect size, normalized. stat. = the test statistic which is used in computation of significance. Symbols are defined as in the previous sections.

Table 7 describes common testing procedures which may both be seen as tests for a guarantee of expected rank difference $r_k - r_{k'}$ in future data scenario (c). The sign test is a binomial test regarding the proportion $p_{k,k'}$ being significantly different from $\frac{1}{2}$. In case of ties, a trinomial test is used. The implemented version of the Friedman test uses the F-statistic (and not the Q-statistic aka chi-squared-statistic) as described in [16].

For post-hoc comparison and visualization of average rank differences, mlaut implements the combination of Bonferroni and studentized rannge multiple testing correction with Neményi's confidence intervals, as described in 3.3.

4 MLaut, API Design and Main Features

MLaut [32] is a modelling and workflow toolbox that was written with the aim of simplifying the task of running machine learning benchmarking experiments. MLaut was created with the specific use-case of large-scale performance evaluation on a large number of real life datasets, such as the study of [19]. Another key goal was to provide a scalable and unified high-level interface to the most important machine learning toolboxes, in particular to include deep learning models in such a large-scale comparison..

Below, we describe package design and functionality. A short usage handbook is included in Section 4.5

MLaut may be obtained from pyPI via `pip install mlaut`, and is maintained on GitHub at `github.com/alan-turing-institute/mlaut`. A Docker container can also be obtained from Docker Hub via `docker pull kazakovv/mlaut`.

4.1 Applications and Use

MLaut main use case is the set-up and execution of supervised (classification and regression) benchmarking experiments. The package currently provides an high-level workflow interface to scikit-learn and keras models, but can easily be extended by the user to incorporate model interfaces from additional toolboxes into the benchmarking workflow.

MLaut automatically creates begin-to-end pipeline for processing data, training machine learning experiments, making predictions and applying statistical quantification methodology to benchmark the performance of the different models.

More precisely, MLaut provides functionality to:

- Automate the entire workflow for large-scale machine learning experiments studies. This includes structuring and transforming the data, selecting the appropriate estimators for the task and data data at hand, tuning the estimators and finally comparing the results.
- Fit data and make predictions by using the prediction strategies as described in 5.4 or by implementing new prediction strategies.
- Evaluate the results of the prediction strategies in a uniform and statistically sound manner.

4.2 High-level Design Principles

We adhered to the high-level API design principles adopted for the scikit-learn project [10]. These are:

1. Consistency.
2. Inspection.
3. Non-proliferation of classes.
4. Composition.
5. Sensible defaults.

We were also inspired by the Weka project [29], a platform widely used for its data mining functionalities. In particular, we wanted to replicate the ease of use of Weka in a pythonic setting.

4.3 Design Requirements

Specific requirements arise from the main use case of scalable benchmarking and the main design principles:

1. **Extensibility.** MLaut needs to provide a uniform and consistent interface to level 3 toolbox interfaces (as in Section 1.1). It needs to be easily extensible, e.g., by a user wanting to add a new custom strategy to benchmark.
2. **Data collection management.** Collections of data sets to benchmark on may be found on the internet or exist on a local computer. MLaut needs to provide abstract functionality for managing such data set collections.

3. **Algorithm/model management.** In order to match algorithms with data sets, MLaut needs to have abstract functionality to do so. This needs to include sensible default settings and easy meta-data inspection of standard methodology.

4. **Orchestration management.** MLaut needs to conduct the benchmarking experiment in a standardized way with minimal user input beyond its specification, with sensible defaults for the experimental set-up. The orchestration module needs to interact with, but be separate from the data and algorithm interface.

5. **User Friendliness.** The package needs to be written in a pythonic way and should not have a steep learning curve. Experiments need to be easy to set-up, conduct, and summarize, from a python console or a jupyter notebook.

In our implementation of MLaut, we attempt to address the above requirements by creating a package which:

- *Has a nice and intuitive scripting interface.* One of our main requirements was to have a native Python scripting interface that integrates well with the rest of our code. Our design attempts to reduce user interaction to the minimally necessary interface points of experiment specification, running of experiments, and querying of results.

- *Provides a high level of abstraction form underlying toolboxes.* Our second criteria was that MLaut provided high level of abstraction from underlying toolboxes. One of our main requirements was for MLaut to be completely model and toolbox agnostic. The scikit-learn interface was too lightweight for our purposes as its parameter and meta-data management is not interface explicit (or inspectable).

- *Provides Scalable workflow automation.* This needed to be one of MLaut's cornerstone contributions. Its main logic is implemented in the `orchestrator` class that orchestrates the evaluation of all estimators on all datasets. The class manages resources for building the estimator models, saving/loading the data and the estimator models. It is also aware of the experiment's partial run state and can be used for easy resuming of an interrupted experiment.

- *Allows for easy estimator construction and retrieval.* The end user of the package should be able to easily add new machine learning models to the suite of build in ones in order to expand its functionality. Besides a small number of required methods to implement, we have provided interfaces to two of the most used level 3 toolbox packages, sklearn and keras.

- *Has a dedicated meta-data interface for sensible defaults of estimators.* We wanted to ensure that the estimators that are packaged in MLaut come with sensible defaults, i.e. pre-defined hyperparameters and tuning strategies that should be applicable in most use cases. The robustness of these defaults has been tested and proven as part of the original large-scale classification study. As such, the user is not required to have a detailed understanding of the algorithms and how they need to be set up, in order to make full use them.

- *Provides a framework for quantitative benchmark reporting.* Easily accessible evaluation methodology for the benchmarking experiments is one of the key features of the package. We also considered reproducibility of results as vital, reflected in a standardized set-up and interface for the experiments, as well as control throughout of pseudo-random seeds..

- *Orchestrates the experiments and parallelizes the load over all available CPU cores.* A large benchmarking study can be quite computationally expensive. Therefore, we needed to make sure that all available machine resources are fully utilized in the process of training the estimators. In order to achieve this we used the parallelization methods that are available as part of the `GridSearch` method and natively with some of the estimators. Furthermore, we also provide a Docker container for running MLaut which we recommend using as a default as it allows the package to run in the background at full load.

- *Provides a uniform way of storing a retrieving data.* Results of benchmarking experiments needed to be saved in a uniform way and made available to users and reviewers of the code. At the current stage, we implemented back-end functionality for management via local HDF5 database files. In the future, we hope to support further data storage back-ends with the same orchestrator-sided facade interface.

4.3.1 Estimator encapsulation

MLaut implements a logic of encapsulating the meta-data with the estimators that it pertains to. This is achieved by using a decorator class that is attached to each estimator class. By doing this, our extended interface is are able to bundle wide-ranging meta-data information with each estimator class. This includes:

- Basic estimator properties such as name, estimator family;
- Types of tasks that a particular estimator can be applied to;
- The type of data which the estimator expects or can handle;
- The model architecture (on level 3, as in Section 1.1). This is particularly useful for more complex estimators such as deep neural networks. By applying the decorator structure the model architecture can be easily altered without changing the underlying estimator class.

This extended design choice has significant benefits for a benchmarking workflow package. First of all, it allows fsearching for estimators based on some basic criteria such as task or estimator family. Second of all, it allows to inspect, query, and change default hyper-parameter settings used by the estimators. Thirdly, strategies with different internal model architectures can be deployed with relative ease.

4.3.2 Workflow design

The workflow supported by MLaut consists of the following main steps:

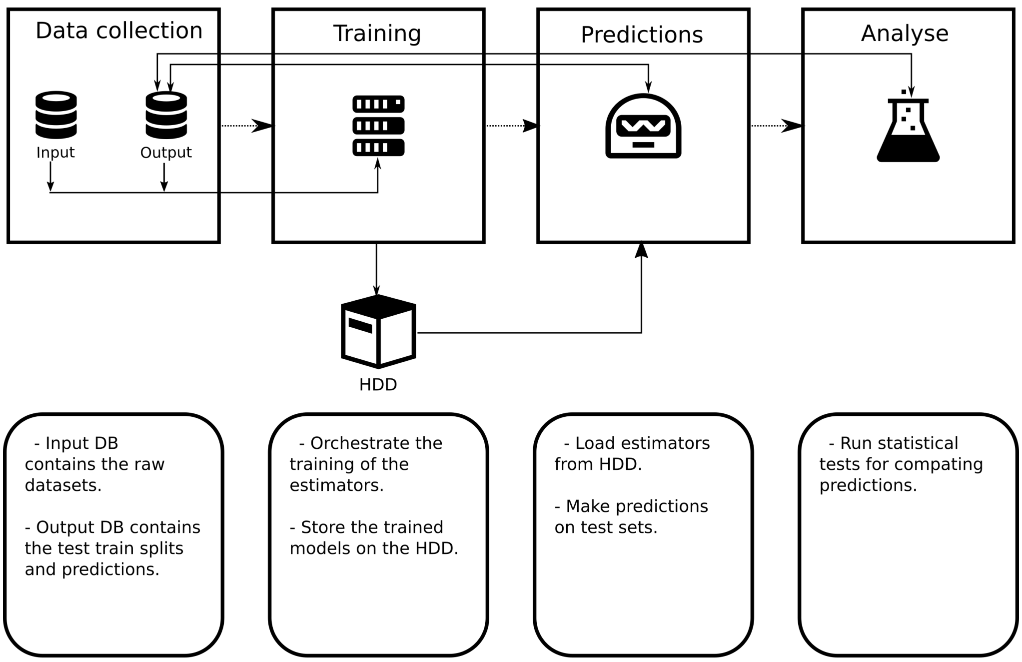

Figure 1: The orchestrated MLaut workflow

1. **Data collection.** As a starting point the user needs to gather and organize the datasets of interest on which the experiments will be run. The raw datasets need to be saved in a HDF5 database. Metadata needs to be attached to each dataset which is later used in the training phase for example for distinguishing the target variables. MLaut provides an interface for manipulating the databases through its `Data` and `Files_IO` classes. The logic of the toolbox is to provision two HDF5 databases one for storing the input data such as the datasets and a second one to store the output of the machine learning experiments and processed data such as train/test index splits. This separation of input and output is not required but is recommended. The datasets also need to be split in a *train* and *test* set in advance of proceeding with the next phase in the pipeline. The indices of the train and test splits are stored separately from the actual datasets in the HDF5 database to ensure data integrity and reproducibility of the experiments. All estimators are trained

and tuned on the training set only. At the end of this process the estimators are used on the test sets which guarantees that all predictions are made on unseen data.

2. **Training phase.** After the datasets are stored in the HDF5 database by following the convention adopted by MLaut the user can proceed to training the estimators. The user needs to provide an array of machine learning estimators that will be used in the training process. MLaut provides a number of default estimators that can be instantiated. This can be done by the use of the `estimators` module. The package also provides the flexibility for the user to write its own estimator by inheriting from the `mlaut_estimator` class. Furthermore, there is a `generic_estimator` module which provides flexibility for the user to create new estimators with only a couple of lines of code.

   The task of training the experiments is performed by the `experiments. Orchestrator` class. This class manages the sequence of the training the the parallelization of the load. Before training each dataset is preprocessed according to metadata provided on the estimator level. This includes normalizing the features and target variables, conversion from categorical to numerical values.

   We recommend running the experiments inside a Docker container if they are very computationally intensive. This allows MLaut to run in the background on a server without shutting down unexpectedly due to loss of connection. We have provided a Docker image that makes this process easy.

3. **Making predictions.** During training the fitted models are stored on the hard drive. At the end of the training phase the user can again use `experiments.Orchestrator` class to retrieve the trained models and make predictions on the test sets.

4. **Analyse results.** The last stage is analysing the output of the results of the machine learning experiments. In order to initiate the process the user needs to call the `analyze_results.prediction_errors` method which returns two dictionaries with the average errors per estimator on all datasets as well as the errors per estimator achieved on each dataset. These results can be used as inputs to the statistical tests that are also provided as part of the `analyze_rezults` module which mostly follow the methodology proposed by [16].

4.4 Software Interface and Main Toolbox Modules

MLaut is built around the logic of the pipeline workflow described earlier. Our aim was to implement the programming logic for each step of the pipeline in a different module. The code that is logically used in more than one of the stages is implemented in a `Shared` module that is accessible by all other classes. The current design pattern is most closely represented by the *façade* and *adaptor* patterns under which the user interacts with one common interface to access the underlying adaptors which represent the underlying machine learning and statistical toolboxes.

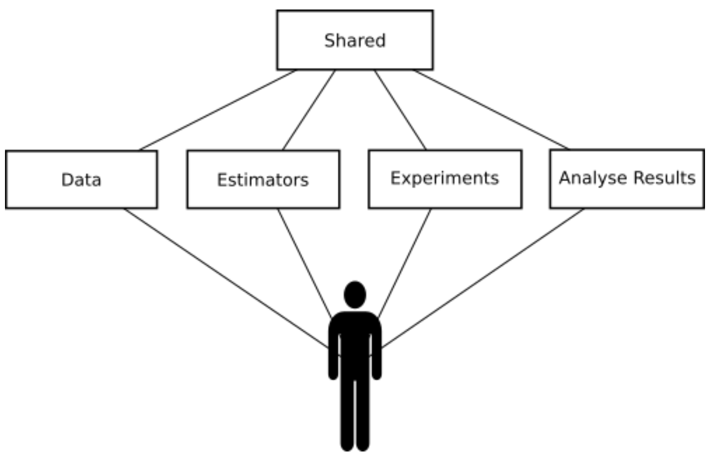

Figure 2: Interaction between the main MLaut modules

4.4.1 Data Module

The `Data` module contains the high level methods for manipulating the raw datasets. It provides a second layer of interface to the lower level classes for accessing, storing and extracting data from HDF5 databases. This module uses heavily the functionality developed in the `Shared` module but provides a higher level of abstraction for the user.

4.4.2 Estimators Module

This module encompasses all machine learning models that come with MLaut as well as methods for instantiating them based on criteria provided by the user. We created MLaut for the purpose of running supervised classification experiments but the toolbox also comes with estimators that can be used for supervised regression tasks.

From a software design perspective the most notable method in this class is the `build` method which returns an instantiated estimator with the the appropriate hyper parameter search space and model architecture. In software design terms this approach resembles more closely the *builder design pattern* which aims at separating the construction of and object from its representation. This design choice allows the base `mlaut_estimator` class to create different representations of machine learning models.

The `mlaut_estimator` object includes methods that complete its set of functionalities. Some of the main ones are a `save` method that takes into account the most appropriate format to persist a trained estimator object. This could include the pickle format used by most *scikit-learn* estimators or the *HDF5* format used by keras. A `load` function is also available for restoring the saved estimators.

The design of the package also relies on the estimators having a uniform `fit` and `predict` methods that takes the same input date and generate predictions in the same format. These methods are not implemented at the `mlaut_estimator` level but instead we relied on the fact that these fundamental methods will be uniform across the underlying packages. However, there is a discrepancy in the behaviour of the *scikit-learn* and *keras* estimators. For classification tasks *keras* requires the labels of the training data to be one hot encoded. Furthermore, the default behaviour of the keras `predict` method is equivalent to the `predict_proba` in scikit-learn. We solved these discrepancies by overriding the `fit` and `predict` methods of the implemented keras estimators.

Through the use of decorators and by implementing the `build` method we are able to fully customize the estimator object with minimal required programming. The decorator class allows to set the metadata associated with the estimator. This includes setting the name, estimator family, types of tasks and hyper parameters. This together with an implemented `build` method will give the user a fully specified machine learning model. This approach also facilitates the application of the algorithms and the use of the software as we can ensure that each algorithm is matched to the correct datasets. Furthermore, this allows to easily retrieve the required algorithms by executing a simple command.

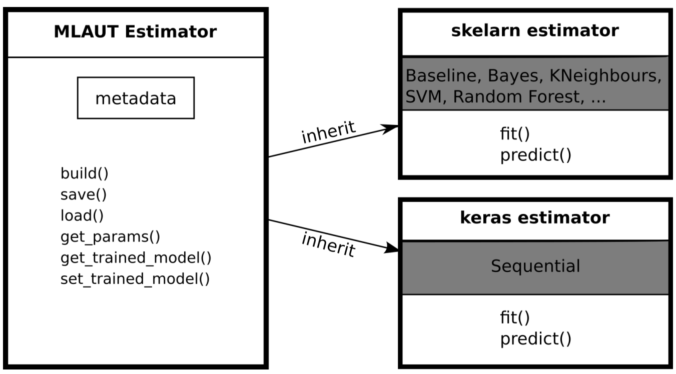

Figure 3: Interaction between MLaut Estimator class and third-party ml estimators

Closely following terminology and taxonomy of [30], mlaut estimators are currently assigned to one of the following methodological families:

a) **Baseline Estimators**. This family of models is also referred to as a dummy estimator and serves as a benchmark to compare other models to. It does not aim to learn any representation of the data but simply adopts a strategy of guessing.

b) **Generalized Linear Model Estimators**. A family of models that assumes that a (generalized) linear relationship exists between the dependent and target values.

c) **Naive Bayes Estimators**. This class of models applies the Bayes theorem my making the naive assumption that all features are independent.

d) **Prototype Method Estimators**. Family of models that apply prototype matching techniques for fitting the data. The most prominent member of this family is the K-means algorithm.

e) **Kernel Method Estimators**. Family of models using kernelization techniques, including support vector machine based estimazors.

f) **Deep Learning and Neural Network Estimators**. This family of models provides implementation of neural network models, including deep neural networks.

g) **Ensembles-of-Trees Estimators**. Family of methods that combines the predictions of several tree-based estimators in order to produce a more robust overall estimator. This family is further divided in:

   (a) *averaging methods*. The models in this group average the predictions of several independent models in order to arrive at a combined estimator. An example is Breiman's random forest.

   (b) *boosting methods*. An ensembling approach of building models sequentially based on iterative weighted residual fitting. An example are stochastic gradient boosted tree models.

In addition ot this the user also has the option to write their own estimator objects. In order to achieve this the new class needs to inherit from the `mlaut_estimator` class. and implementing the abstract methods in each child class. The main abstract method that needs to be implemented is the `build` method which returns an wrapped instance of the estimator with a set of hyper-parameters that will be used in the tuning process. For further details about the implemented estimators refer to 5.4.

4.4.3 Experiments Module

This module contains the logic for orchestration of the machine learning experiments. The main parameters in this module are the datasets and the estimator models that will be trained on the data. The main `run` method of the module then proceeds to training all estimators on all datasets, sequentially. The core of the method represent two embedded `for` loops the first of which iterates over the datasets and the second one over the estimators. Inside the inner loop the orchestrator class builds an estimator instance for each dataset. This allows to tailor the machine learning model for each dataset. For example, the architecture of a deep neural network can be altered to include the appropriate number of neurons based on the input dimensions of the data. This module is also responsible for saving the trained estimators and making predictions. It should be noted that the orchestrator module is not responsible for the parallelization of the experiments which is handled on an individual estimator level.

4.4.4 Result Analysis Module

This module includes the logic for performing the quantitative evaluation and comparison of the machine learning strategies' performance. The predictions of the trained estimators on the test sets for each dataset serve as input. First, performances and, if applicable, standard errors on the individual data sets are computed, for a given average or aggregate loss/performance quantifier. The samples of performances are then used as inputs for comparative quantification.

API-wise, the framework for assessing the performance of the machine learning estimators hinges on three main classes. The `anlyze_results` class implements the calculation of the quantifiers. Through composition this class relies on the `losses` class that performs the actual calculation of the prediction performances over the individual test sets. The third main class that completes the framework design is the `scores` class. It defines the loss/quantifier function that is used for assessing the predictive power of the estimators. An instance of the `scores` class is passed as an argument to the `losses` class.

We believe that this design choice of using three classes is required to provide the necessary flexibility for the composite performance quantifiers as described in Section 3 - i.e., to allow to compute ranks for an arbitrarily chosen loss (e.g., mean rank with resect to mean absolute error), or to perform comparison testing using an arbitrarily chosen performance quantifier (e.g., Wilcoxon signed rank test comparing F1-scores).

Our API also facilitates user custom extension, e.g., for users who wish to add a new score function, an efficient way to compute aggregate scores or standard errors, or a new comparison testing methodology. For example, adding new score functions can be easily achieved by inheriting from the `MLautScore` abstract base class. On the other hand, the `losses` class completely encapsulates the logic for the calculation of the predictive performance of the estimators. This is particularly useful as the class internally implements a mini orchestrator procedure for calculating and presenting the loss achieved by all estimators supplied as inputs. Lastly, the suite of statistical tests available in MLaut can be easily expanded by adding the appropriate method to the `analyze_results` class or a descendant.

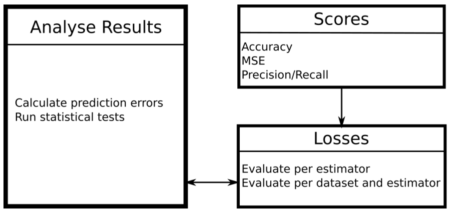

Figure 4: Interaction between main classes inside the `analyse_results` module

Mathematical details of the implemented quantification procedures implemented in MLaut were presented in Section 3. Usage details

In this implementation of MLaut we use third-party packages for performing the statistical tests. We rely mostly on the `scikit-learn` package. However, for post hoc tests we use the `scikit-posthocs` package [43] and the `Orange` package [17] which we also used for creating critical distance graphs for comparing multiple classifiers.

4.4.5 Shared Module

This module includes classes and methods that are shared by the other modules in the package. The `Files_IO class` comprises of all methods for manipulating files and datasets. This includes saving/loading of trained estimators from the HDD and manipulating the HDF5 databases. The `Shared` module also keeps all static variables that are used throughout the package.

4.5 Workflow Example

We give a step-by-step overview over the most basic variant of the user workflow. Advanced examples with custom estimators and set-ups may be found in the MLaut tutorial [32].

Step 0: setting up the data set collection

The user should begin by setting up the data set collection via the `Files_IO class`. Meta-data for each dataset needs to be provided that includes as a minimum the class column name/target attribute and name of dataset. This needs to be done *o*nce for every dataset collection, and may not need to be done for a pre-existing or pre-deployed collection. Currently, only local HDF5 data bases are supported.

We have implemented back-end set-up routines which download specific data set collections and generate the meta-data automatically. Current support includes the UCI library data sets and OpenML. Alternatively, the back-end may be populated directly by storing an in-memory pandas DataFrame via the `save_pandas_dataset` method, e.g., as part of custom loading scripts.

```
input_io.save_pandas_dataset(
        dataset=data, #in Pandas DataFrame format
        save_loc='/openml', #location in HDF5 database
        metadata=metadata)
```

In this case, meta-data for the individual datasets needs to be provided in the following dictionary format:

```
metadata = {
        'class_name': ..., #label column name
```

```
        'source': , # source of the data
        'dataset_name': ... ,
        'dataset_id': id
        }
```

Step 1: initializing data and output locations

As the next step, the user should specify the back-end links to the data set collections ("input") and to intermediate or analysis results ("output").This is done via the `data` class. It is helpful for code readability to store these in codeinput_io and `out_io` variables.

```
input_io = data.open_hdf5('data/openml.h5', mode='r')
out_io = data.open_hdf5(
        'data/openml-classification.h5', mode='a')
```

These may then be supplied as parameters to preparation and orchestration routines. We then proceed to getting the paths to the raw datasets as well as the respective train/test splits which is performed respectively though the use of `list_datasets` and `split_datasets` methods.

```
(dts_names_list,
dts_names_list_full_path) = data.list_datasets(
        hdf5_io=input_io,
        hdf5_group='openml/')

split_dts_list = data.split_datasets(hdf5_in=input_io,
        hdf5_out=out_io,
        dataset_paths=dts_names_list_full_path)
```

Step 2: initializing estimators

The next step is to instantiate the learning strategies, estimators in sklearn terminology, which we want to use in the benchmarking exercise. The most basic and fully automated variant is use of the `instantiate_default_estimators` method which loads a pre-defined set of defaults given specified criteria. Currently, only a simple string look-up via the `estimators` parameter is implemented, but we plan to extend the search/matching functionality. The string criterion may be used to fetch specific estimators by a list of names, entire families of models, estimators by task (e.g., classification), or simply all available estimators.

```
instantiated_models = instantiate_default_estimators(
        estimators=['Classification'],
        verbose=1,
        n_jobs=-1)
```

Step 3: orchestrating the experiment

The final step is to run the experiment by passing references to data and estimators to the `orchestrator` class, then initating the training process by invoking its `run` method.

```
orchest = orchestrator(hdf5_input_io=input_io,
        hdf5_output_io=out_io,
        dts_names=dts_names_list,
        original_datasets_group_h5_path='openml/')

orchest.run(modelling_strategies=instantiated_models)
```

Step 4: computing benchmark quantifiers

After the estimators are trained and the predictions of the estimators are recorded we can proceed to obtaining quantitative benchmark results for the experiments.

For this, we need to instantiate the `AnalyseResults` class by supplying the folders where the raw datasets and predictions are stored. Its `prediction_errors` method may be invoked to returns both the calculated prediction performance quantifiers, per estimator as well as the prediction performances per estimator and per dataset.

```
analyse = AnalyseResults(
        hdf5_output_io=out_io,
        hdf5_input_io=input_io,
        input_h5_original_datasets_group='openml/',
        output_h5_predictions_group='experiments/predictions/')

#Score function that will be used for the statistical tests
score_accuracy = ScoreAccuracy()

estimators = instantiate_default_estimators(['Classification'])

(errors_per_estimator,
errors_per_dataset_per_estimator,
errors_per_dataset_per_estimator_df) =
        analyse.prediction_errors(metric=score_accuracy, estimators=estimators)
```

The prediction errors per dataset per estimator can be directly examined by the user. On the other hand, the estimator performances may be used as further inputs for comparative quantification via hypothesis tests. For example, we can perform a paired t-test for pairwise comparison of methods by invoking the code below:

```
t_test, t_test_df = analyze.t_test(errors_per_estimator)
```

5 Using MLaut to Compare the Performance of Classification Algorithms

As an major test use case for MLaut, we conducted a large-scale benchmark experiment comparing a selection of off-shelf classifiers on datasets from the UCI Machine Learning Repository. Our study had four main aims:

(1) stress testing the MLaut framework on scale, and observing the user interaction workflow in a major test case.
(2) replicating the key points of the experimental set-up by [19], while avoiding their severe mistake of tuning on the test set.
(3) including deep learning methodology to the experiment.

Given the above, the below benchmarking study is, to the best of our knowledge, the first large-scale supervised classification study which[3]:

(a) is correctly conducted via out-of-sample evaluation and comparison. This is since [19] commit the mistake of tuning on the test set, as it is even acknowledged in their own Section 3 Results and Discussion.
(b) includes contemporary deep neural network classification approaches, and is conducted on a broad selection of classification data sets which is not specific to a special domain such as image classification (the UCI dataset collection).

We intend to extend the experiment in the future by including further dataset collections and learning strategies.

Full code for our experiments, including random seeds, can be found as a jupyter notebook in MLaut's documentation [32].

[3]in the disjunctive sense: i.e., to the best of our knowledge, the first large-scale benchmarking study which does *any* of the above rather than being only the first study to do *all* of the above.

5.1 Hardware and software set-up

The benchmark experiment was conducted on a Microsoft Azure VM with 16 CPU cores and 32 GB of RAM, by our Docker virtualized implementation of MLaut. The experiments ran for about 8 days. MLaut requires Python 3.6 and should be installed in a dedicated virtual environment in order to avoid conflicts or the Docker implementation should be used. The full code for running the experiments and the code for generating the results in results Appendix A can be found in the examples directory in the GitHub repository of the project.

5.2 Experimental set-up

5.2.1 Data set collection

The benchmarking study uses the same dataset collection as employed by [19]. This collection consists of 121 tabular datasets for supervised classification, taken directly from the UCI machine learning repository. Prior to the experiment, each dataset was standardized, such that each individual feature variable has a mean of 0 and a standard deviation of 1.

The dataset collection of [19] intends to be representative of a wide scope of basic real-world classification problems. It should be noted that this representative cross-section of simple classification tasks *excludes* more specialized tasks such as image, audio, or text/document classification which are usually regarded to by typical applications of deep learning, and for which deep learning is also the contemporary state-of-art. For a detail description of the Fernández-Delgado et al. [19] data collection, see section 2.1 there.

5.2.2 Re-sampling for evaluation

Each dataset is in split into exactly one pair of training and test set. The training sets are selected, for each data set, uniformly at random[4] as (a rounded) $\frac{2}{3}$ of the available data sample; the remaining $\frac{1}{3}$ in the dataset form the test set on which the strategies are asked to make predictions. Random seeds and the indices of the exact splits were saved to ensure reproducibility and post-hoc scrutiny of the experiments.

The training set may (or may not) be further split by the contender methods for tuning - as stated previously in Section 2, this is not enforced as part of the experimental set-up[5], but is left to each learning strategy to deal with internally, and will be discussed in the next section. In particular, none of the strategies have access to the test set for tuning or training.

5.3 Evaluation and comparison

We largely followed the procedure suggested by [16] for the analysis of the performance of the trained estimators. For all classification strategies, the following performance quantifiers are computed per dataset:

1. misclassification loss
2. rank of misclassification loss
3. runtime

Averages of these are computed, with standard errors for future data situation (c: re-trained, on unseen dataset). In addition, for the misclassification loss on each data set, standard errors for future data situation (a: re-used, same dataset) are computed.

The following pairwise comparisons between samples of performances by dataset are computed:

1. paired t-test on misclassification losses, with Bonferroni correction
2. (paired) Wilcoxon signed rank on misclassification losses, with Bonferroni correction
3. Friedman test on ranks, with Neményi's significant rank differences and post-hoc significances

Detail descriptions of these may be found in Section 3.

[4] independently for each dataset in the collection
[5] Unlike in the set-up of [19] which, on top of doing so, is also faulty.

5.4 Benchmarked machine learning strategies

Our choice of classification strategies is not exhaustive, but is meant to be representative of off-shelf choices in the *scikit-learn* and *keras* packages. We intend to extend the selection in future iterations of this study.

From *scikit-learn*, the suite of standard off-shelf approaches includes linear models, Naive Bayes, SVM, ensemble methods, and prototype methods.

We used *keras* to construct a number of neural network architectures representative of the state-of-art. This proved a challenging task due to the lack of explicitly recommended architectures for simple supervised classification to be found in literature.

5.4.1 Tuning of estimators

It is important to note that the off-shelf choices and their default parameter settings are often not considered good or state-of-art: hyper-parameters in *scikit-learn* are by default not tuned, and there are no default *keras* that come with the package.

For *scikit-learn* classifiers, we tune parameters using *scikit-learn*'s `GridSearchCV` wrapper-compositor (which never looks at the test set by construction).

In all cases of tuned methods, parameter selection in the inner tuning loop is done via grid tuning by 5-fold cross-validation, with respect to the default *score* function implemented at the estimator level. For classifiers as in our study, the default tuning score is mean accuracy (averaged over all 5 tuning test folds in the inner cross-validation tuning loop), which is equivalent to tuning by mean misclassification loss.

The tuning grids will be specified in Section 5.4.2 below.

For *keras* classifiers, we built architectures by interpolating general best practice recommendations in scientific literature [34], as well as based on concrete designs found in software documentation or unpublished case studies circulating on the web. We further followed the sensible default choices of *keras* whenever possible.

The specific choices for neural network architecture and hyper-parameters are specified in Section 5.4.3 below.

5.4.2 Off-shelf scikit-learn supervised strategies

i) Algorithms that do not have any tunable hyperparameters

Estimator name	**`sklearn.dummy.DummyClassifier`**
Description	This classifier is a naive/uninformed baseline and always predicts the most frequent class in the training set ("majority class"). This corresponds to the choice of the *most_frequent* parameter.
Hyperparameters	None
Estimator name	**`sklearn.naive_bayes.BernoulliNB`**
Description	Naive Bayes classifier for multivariate Bernoulli models. This classifier assumes that all features are binary, if not they are converted to binary. For reference please see [7] Chapter 2.
Hyperparameters	None
Estimator name	**`sklearn.naive_bayes.GaussianNB`**
Description	Standard implementation of the Naive Bayes algorithm with the assumption that the features are Gaussian. For reference please see [7] Chapter 2.
Hyperparameters	None

ii) Linear models

Estimator name	**`sklearn.linear_model.PassiveAggressiveClassifier`**
Description	Part of the online learning family of models based on the *hinge loss* function. This algorithm observes feature-value pairs z in sequential manner. After each observation the algorithm makes a prediction, checks the correct value and calibrates the weights. For further reference see [15].
Hyperparameters	`C`: array of 13 equally spaced numbers on a log scale in the range $[10^{-2}; 10^{10}]$ `scikit-learn` default: 1

iii) Clustering Algorithms

Estimator name **`sklearn.neighbors.KNeighborsClassifier`**

Description The algorithm uses a majority vote of the nearest neighbours of each data point to make a classification decision. For reference see [14] and [7], Chapter 2.

Hyperparameters `n_neighbors=[1;30]`, `scikit-learn` default: 5
`p=[1,2], scikit-learn default:2`

iv) Kernel Methods

Estimator name **`sklearn.svm.SVC`**

Description This estimator is part of the Support Vector family of algorithms. In this study, we use the Gaussian kernel only. For reference see [13] and [7], Chapter 7. The performance of support vector machine is very sensitive with respect to tuning parameters:

- **C**, the regularization parameter. There does not seem to be a consensus in the community regarding the space for the C hyper-parameter search. In an example[6] the `scikit-learn` documentation refers to an initial hyper-parameter search space for C in the range $[10^{-2}; 10^{10}]$ [39]. However, a different example[7] suggests $[1, 10, 100, 1000]$. A third `scikit-learn` example[8] suggests testing for both the linear and rbf kernels and broad values for the C and γ parameters. Other researches [27] suggest to use apply a search for C in the range $[2^{-5}; 2^{15}]$ which we used in our study as it provides a good compromise between reasonable running time and comprehensiveness of the search space.
- γ, the inverse kernel bandwith. The `scikit-learn` example[9] [39] suggests hyper-parameter search space for γ in the range $[10^{-9}; 10^{3}]$. However, a second `scikit-learn` example[10] suggest to search only in $[0.0001, 0.001]$. On the other hand, [27] suggest searching for γ in the range $[2^{-15}; 2^{3}]$ which again we found to be the middle ground and applied in our study.

Hyperparameters C: array of 13 equally spaced numbers on a log scale in the range $[2^{-5}; 2^{15}]$, `scikit-learn` default: 1.
gamma: array of 13 equally spaced numbers on a log scale in the range $[2^{-15}; 2^{3}]$, `scikit-learn` default: auto

v) Ensemble Methods

The three main models that we used in this study that are part of this family are the RandomForest, Bagging and Boosting. The three models are built around the logic of using the predictions of a large number of weak estimators, such as decision trees. As such they share a lot of the same hyperparameters. Namely, some of the main parameters for this family of models are the number of estimators, max number of features and the maximum tree depth, default values for each estimator which are suggested in the `scikit-learn` package. Recent research [36] and informal consensus in the community suggest that the performance gains from deviating from the default parameters are rewarded for the Boosting algorithm but tend to have limited improvements for the RandomForest algorithm. As such, for the purposes of this study we will focus our efforts to tune the Boosting and Bagging algorithms but will use a relatively small parameter search space for tuning RandomForest.

Estimator name	**`sklearn.ensemble.GradientBoostingClassifier`**
Description	Part of the ensemble meta-estimators family of models. We used the default sklearn *deviance* loss. The algorithm fits a series of decision trees on the data and predictions are made based on a majority vote. At each iteration the data is modified by applying weights to it and predictions are made again. At each iteration the weights of the incorrectly x, y pairs are increased (boosted) and decreased for the correctly predicted pairs. As per the `scikit-learn` documentation [39] This estimator is not recommended for datasets with more than two classes as it requires the introduction of regression tress at each iteration. The suggested approach is to use the RandomForest algorithm instead. A lot of the datasets used in this study are multiclass supervised learning problems. However, for the purposes of this study we will use the Gradient Boosting algorithm in order to see how it performs when benchmarked to the suggested approach. For reference see [21] and [18], Chapter 17.
Hyperparameters	number of estimators: $[10, 50, 100]$, `scikit-learn` default: 100. max depth: integers in the range $[1; 10]$, `scikit-learn` default: 3.

Estimator name	**`sklearn.ensemble.RandomForestClassifier`**
Description	Part of the ensemble meta-estimators family of models. The algorithm fits decision trees on sub-samples of the dataset. The average voting rule is used for making predictions. For reference see [9] and [18], Chapter 17.
Hyperparameters	For this study we used the following hyperparameter grid: number of estimators: $[10, 50, 100]$, `scikit-learn` default: 10 max features: [auto, sqrt, log2, None], `scikit-learn` default: auto max depth: [10, 100, None], `scikit-learn` default: None
Estimator name	**`sklearn.ensemble.BaggingClassifier`**
Description	Part of the ensemble meta-estimators family of models. The algorithm draws with replacement feature-label pairs (x, y), trains decision base tree estimators and makes predictions based on voting or averaging rules. For reference see [8] and [18], Chapter 17.
Hyperparameters	number of estimators: $[10, 50, 100]$, `scikit-learn` default: 10

5.4.3 Keras neural network architectures including deep neural networks

We briefly summarize our choices for hyper-parameters and architecture.

- **Architecture**. Efforts have been made to make the choice of architectures less arbitrary by suggesting algorithms for finding the optimal neural network architecture [23]. Other researches have suggested good starting points and best practices that one should adhere to when devising a network architecture [25, 38]. We also followed the guidelines of [22], in particular Chapter 6.4. The authors conducted an empirical study and showed that the accuracy of networks increases as the number of layers grow but the gains are diminishing rapidly beyond 5 layers. These findings are also confirmed by other studies [3] that question to need to use very deep feed-forward networks. In general, the consensus in the community seems to be that 2-4 hidden layers are sufficient for most feed-forward network architectures. One notable exception to this rule seem to be convolutional network architecture which have been showed to perform best when several sequential layers are stacked one after the other. However, this study does *not* make use of convolutional neural networks, as our data is not suitable for these models, in particular because there is no well-specified way to transform samples into a multidimensional array form. The architectures are given below as their keras code specification.

- **Activation function**. We used the rectified linear unit (ReLu) as our default choice of activation function as has been found to accelerate convergence and is relatively inexpensive to perform [33].

- **Regularization**. We employ the current state-of-art in neural network regularization: dropout. In the absence of clear rules when and where dropout should be applied, we include two versions of each neural network in the study: one version *not* using dropout, and one using dropout. Dropout regularization is as described by Hinton et al. [26], Srivastava et al. [42] where its potential for improving the generalization accuracy of neural networks is shown. We used a dropout rate of 0.5 as suggested by the authors.

- **Hyper-parameter tuning**. We did not perform grid search to find the optimal hyper parameters for the network. The reason for this is two-fold. We interfaced the neural network models from *keras*. The *keras* interface is not fully compatible with *scikit learn's GridSearch*, nor does it provide easy off-shelf tuning facilities (see subsection 4.4.2 for details). Furthermore, using grid search tuning does not seem to be considered common practice by the community, and it is even actively recommended to avoid by some researchers [5], hence might not be considered a fair representation of the state-of-art. Instead, the prevalent practice seems to be manual tuning of hyper-parameters based on learning curves. Following the latter in the absence of off-shelf automation, we manually tuned learning rate, batch size, and number of epochs by manual inspection of learning curves and performances on the full *training sets* (see below).

- **Learning Rate**. The learning rate is one of the crucial hyper-parameter choices when training neural networks. The generally accepted rule to find the optimal rate is to start with a large rate and if the training process does not diverge decrease the learning rate by a factor of 3 [4]. This approach is confirmed by [42] who also affirm that a larger learning rate can be used in conjunction with dropout without risking that the weights of the model blow out.

- **Batch Size**. The datasets used in the study were relatively small and could fit in the memory of the machine that we used for training the algorithms. As a result we set the batch size to equal the entire dataset which is equivalent to full gradient descent.

- **Number of epochs**. We performed manual hyper-parameter selection by inspection of individual learning curves for all combinations of learning rate and architecture. For this, learning curves on individual data sets' training samples were inspected visually for the "plateau range" (range of minimal training error). For all architectures, and most data sets, the plateau was already reached for *one single epoch*, and training error usually tended to increase in the range of 50-500 epochs. The remaining, small number of datasets (most of which were of 4-or-above-digit sample size) plateaued in the 1-digit range.
  While this is a *very surprising finding* as it corresponds to a single gradient descent step, *it is what we found*, while following what we consider the standard manual tuning steps for neural networks. We further discuss this in Section 5.6 and acknowledge that *this surprising finding warrants further investigation*, e.g., through checking for mistakes, or including neural networks tuned by automated schemes.
  Thus, all neural networks architectures were trained for *one single epoch* - since choosing a larger (and more intuitive number of epochs) would have been somewhat arbitrary, and not in concordance with the common manual tuning protocol.

For the keras models, we adopted six neural network architectures with varying depths and widths. Our literature review revealed that there is no consistent body of knowledge or concrete rules pertaining to constructing neural network models for simple supervised classification (as opposed to image recognition etc). Therefore, we extrapolated from general best practice guidelines as applicable to our study, and also included (shallow) network architectures that were previously used in benchmark studies. The full *keras* architecture of the neural networks used are listed below.

Estimator name **`keras.models.Sequential`**

Description Own architecture of Deep Neural Network model applying the principles highlighted above. For this experiment we made used of the empirical evidence that networks of 3-4 layers were sufficient to learn any function discussed in [3]. However, we opted for a slightly narrower network in order to investigate whether wider nets tend to perform better than narrow ones.

code_examples/deep_nn_4_layer_thin_dropout.py

```
model = OverwrittenSequentialClassifier()
model.add(Dense(288, input_dim=input_dim, activation='relu'))
model.add(Dense(144, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(12, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model_optimizer = optimizers.Adam(lr=lr)
model.compile(loss='mean_squared_error', optimizer=
    model_optimizer, metrics=['accuracy'])
```

Hyperparameters batch size: None, learning rate: $[1, 0.01, 0.001]$, loss: mean squared error, optimizer: Adam, metrics: accuracy.

Estimator name **`keras.models.Sequential`**

Description In this architecture we experimented with the idea that wider networks perform better than narrower ones. No dropout was performed in order to test the idea that regularization is necessary for all deep neural network models.

code_examples/deep_nn_4_layer_wide_no_dropout.py

```
nn_deep_model = OverwrittenSequentialClassifier()
nn_deep_model.add(Dense(2500, input_dim=input_dim, activation='
    relu'))
nn_deep_model.add(Dense(2000, activation='relu'))
nn_deep_model.add(Dense(1500, activation='relu'))
nn_deep_model.add(Dense(num_classes, activation='softmax'))

model_optimizer = optimizers.Adam(lr=lr)
nn_deep_model.compile(loss='mean_squared_error', optimizer=
    model_optimizer, metrics=['accuracy'])
```

Hyperparameters batch size: None, learning rate: $[1, 0.01, 0.001]$, loss: mean squared error, optimizer: Adam, metrics: accuracy.

Estimator name **`keras.models.Sequential`**

Description We tested the same architecture as above but applying dropout after the first two layers.

code_examples/deep_nn_4_layer_wide_with_dropout.py

```
nn_deep_model = OverwrittenSequentialClassifier()
nn_deep_model.add(Dense(2500, input_dim=input_dim, activation='
    relu'))
nn_deep_model.add(Dense(2000, activation='relu'))
nn_deep_model.add(Dropout(0.5))
nn_deep_model.add(Dense(1500, activation='relu'))
nn_deep_model.add(Dense(num_classes, activation='softmax'))

model_optimizer = optimizers.Adam(lr=lr)
nn_deep_model.compile(loss='mean_squared_error', optimizer=
    model_optimizer, metrics=['accuracy'])
```

Hyperparameters batch size: None, learning $[1, 0.01, 0.001]$, loss: mean squared error, optimizer: Adam, metrics: accuracy.

Estimator name **`keras.models.Sequential`**

Description Deep Neural Network model inspired from architecture suggested by [38]:

code_examples/deep_nn_12_layer_wide_with_dropout.py

```
nn_deep_model = OverwrittenSequentialClassifier()
nn_deep_model.add(Dense(5000, input_dim=input_dim, activation='relu'))
nn_deep_model.add(Dense(4500, activation='relu'))
nn_deep_model.add(Dense(4000, activation='relu'))
nn_deep_model.add(Dropout(0.5))

nn_deep_model.add(Dense(3500, activation='relu'))
nn_deep_model.add(Dense(3000, activation='relu'))
nn_deep_model.add(Dense(2500, activation='relu'))
nn_deep_model.add(Dropout(0.5))


nn_deep_model.add(Dense(2000, activation='relu'))
nn_deep_model.add(Dense(1500, activation='relu'))
nn_deep_model.add(Dense(1000, activation='relu'))
nn_deep_model.add(Dropout(0.5))

nn_deep_model.add(Dense(500, activation='relu'))
nn_deep_model.add(Dense(250, activation='relu'))
nn_deep_model.add(Dense(num_classes, activation='softmax'))

model_optimizer = optimizers.Adam(lr=lr)
nn_deep_model.compile(loss='mean_squared_error', optimizer=model_optimizer, metrics=['accuracy'])
```

Hyperparameters batch size: None, learning rate: $[1, 0.01, 0.001]$, loss: mean squared error, optimizer: Adam, metrics: accuracy.

Estimator name **`keras.models.Sequential`**

Description Deep Neural Network model suggested in [26] with the following architecture:

code_examples/keras_nn_4_layer_wide_dropout_each_layer.py

```
nn_deep_model = OverwrittenSequentialClassifier()
nn_deep_model.add(Dense(2000, input_dim=input_dim, activation='relu'))
nn_deep_model.add(Dropout(0.5))
nn_deep_model.add(Dense(1000, activation='relu'))
nn_deep_model.add(Dropout(0.5))
nn_deep_model.add(Dense(1000, activation='relu'))
nn_deep_model.add(Dropout(0.5))
nn_deep_model.add(Dense(50, activation='relu'))
nn_deep_model.add(Dropout(0.5))

nn_deep_model.add(Dense(num_classes, activation='softmax'))

model_optimizer = optimizers.Adam(lr=lr)
nn_deep_model.compile(loss='mean_squared_error', optimizer=model_optimizer, metrics=['accuracy'])
```

Hyperparameters batch size: None, learning rate: $[1, 0.01, 0.001]$, loss: mean squared error, optimizer: Adam, metrics: accuracy.

Estimator name **keras.models.Sequential**

Description Deep Neural Network model suggested in [42] with the following architecture:

code_examples/deep_nn_2_layer_dropout_input_layer.py

```
nn_deep_model = OverwrittenSequentialClassifier()
nn_deep_model.add(Dropout(0.7, input_shape=(input_dim,)))
nn_deep_model.add(Dense(1024, activation='relu'))
nn_deep_model.add(Dropout(0.5))
nn_deep_model.add(Dense(num_classes, activation='softmax'))

model_optimizer = optimizers.Adam(lr=lr)
nn_deep_model.compile(loss='mean_squared_error', optimizer=
    model_optimizer, metrics=['accuracy'])
```

Hyperparameters batch size: None, learning rate: $[1, 0.01, 0.001]$, loss: mean squared error, optimizer: Adam, metrics: accuracy.

5.5 Results

Table 8 shows an summary overview of results.

	avg_rank	avg_score	std_error	avg training time (in sec)
RandomForestClassifier	4.3	0.831	0.013	14.277
SVC	5.0	0.818	0.014	1742.466
K_Neighbours	5.6	0.805	0.014	107.796
BaggingClassifier	5.8	0.820	0.014	5.231
GradientBoostingClassifier	7.6	0.790	0.016	49.509
PassiveAggressiveClassifier	8.5	0.758	0.016	19.352
NN-4-layer_wide_with_dropout_lr001	10.0	0.692	0.021	14.617
NN-4-layer_wide_no_dropout_lr001	10.5	0.694	0.021	14.609
BernoulliNaiveBayes	10.9	0.707	0.015	0.005
NN-4-layer-droput-each-layer_lr0001	11.2	0.662	0.022	6.786
NN-4-layer_thin_dropout_lr001	11.6	0.652	0.022	2.869
NN-2-layer-droput-input-layer_lr001	11.7	0.655	0.021	5.420
GaussianNaiveBayes	13.4	0.674	0.019	0.004
NN-12-layer_wide_with_dropout_lr001	16.3	0.535	0.023	40.003
NN-2-layer-droput-input-layer_lr01	17.3	0.543	0.023	5.413
NN-2-layer-droput-input-layer_lr1	17.9	0.509	0.023	5.437
NN-4-layer_thin_dropout_lr01	18.0	0.494	0.024	5.559
NN-4-layer_wide_no_dropout_lr01	18.4	0.494	0.022	10.530
NN-4-layer-droput-each-layer_lr1	18.5	0.488	0.022	6.901
NN-4-layer_wide_with_dropout_lr1	18.5	0.483	0.022	10.738
NN-4-layer_wide_no_dropout_lr1	18.6	0.490	0.022	10.561
NN-4-layer_wide_with_dropout_lr01	18.7	0.478	0.022	10.696
NN-4-layer-droput-each-layer_lr01	18.8	0.482	0.022	6.818
NN-12-layer_wide_with_dropout_lr01	18.8	0.479	0.022	70.574
NN-12-layer_wide_with_dropout_lr1	19.0	0.458	0.023	68.505
NN-4-layer_thin_dropout_lr1	19.4	0.462	0.023	4.299
BaselineClassifier	23.7	0.419	0.019	0.001

Table 8: Columns are: average rank (lower is better), classification accuracy, standard error of average score (version c) and training time of the prediction strategy. Performances are estimated as described in Section 5.2. Rows correspond to prediction strategies, increasingly ordered by their average rank. Naming for sklearn estimators is as in Section 5.4.2. Naming of keras estimators is as in Section 5.4.3, followed by a string `dropout` or `no_dropout` indicating whether dropout was used, and by a string `lr` and some number indicating the choice of learning rate.

Figure 5 summarizes the samples of performances in terms of classification accuracy. The sample is performance by method, ranging over data sets, averaged over the test sample within each dataset - i.e., the size of the sample of performance equals the number of data sets in the collection.

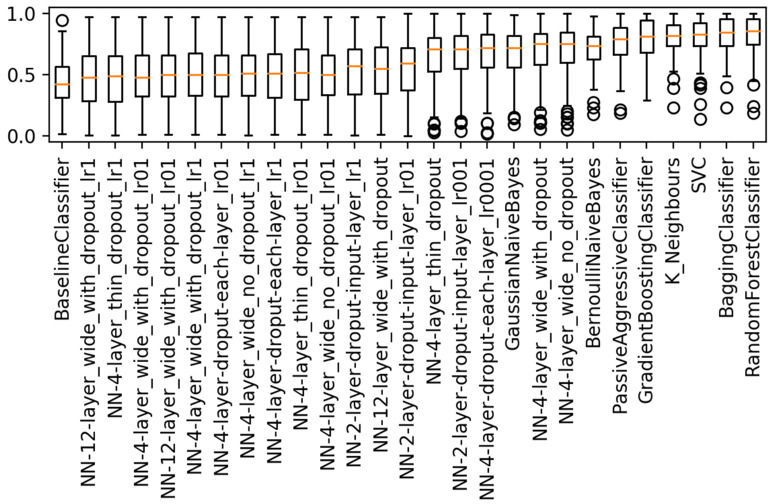

Figure 5: Box-and-whiskers plot of samples of classification accuracy performances classification accuracy by method, ranging over data sets, averaged over the test sample within each dataset. y-axis is classification accuracy. x-axis correspond to prediction strategies, ordered by mean classification accuracy. Naming for sklearn estimators is as in Section 5.4.2. Naming of keras estimators is as in Section 5.4.3, followed by a string `dropout` or `no_dropout` indicating whether dropout was used, and by a string `lr` and some number indicating the choice of learning rate. Whisker length is limited at 1.5 times interquartile range.

The Friedman test was significant at level p=2e-16. Figure 6 displays effect sizes, i.e., average ranks with Neményi's post-hoc critical differences.

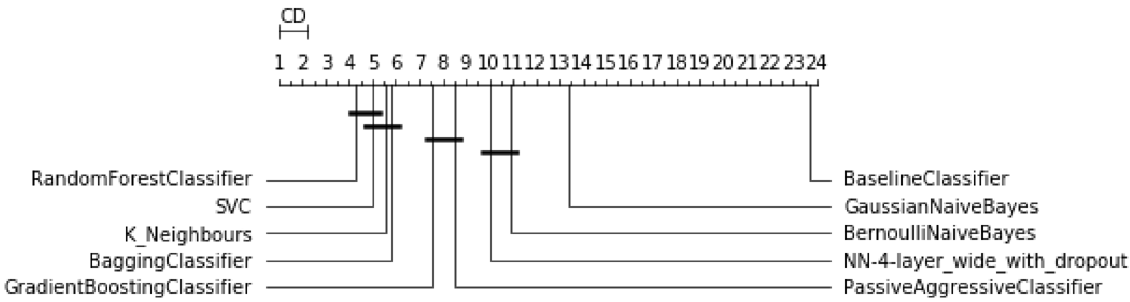

Figure 6: Neményi post-hoc critical differences comparison diagram after [16]. CD = critical average rank difference range. x-axis displays average rank (lower is better). Indicated x-axis location = average ranks of prediction strategy, with strategies of below-critical average rank difference connected by a bar. Naming for sklearn estimators is as in Section 5.4.2. Naming of keras estimators is as in Section 5.4.3, followed by a string `dropout` or `no_dropout` indicating whether dropout was used, and by a string `lr` and some number indicating the choice of learning rate. Note that for the sake of readability, the worst performing neural networks were removed from the plot.

From all the above, the top five algorithms among the contenders were the Random Forest, SVC, Bagging, K Neighbours and Gradient Boosting classifiers.

Further benchmarking results may be found in the automatically generated Appendix A. These include results of paired t-tests and Wilcoxon signed rank tests. Briefly summarizing these: Neither t-test (Appendix A.1), nor the Wilcoxon signed rank test (Appendix A.2), with Bonferroni correction (adjacent strategies and all vs baseline), in isolation, are able to reject the null hypothesis of a performance difference between any two of the top five performers.

5.6 Discussion

We discuss our findings below, including a comparison with the benchmarking study by Fernández-Delgado et al. [19].

5.6.1 Key findings

In summary, the key findings of the benchmarking study are:

(i) MLaut is capable of carrying out large-scale benchmarking experiments across a representative selection of off-shelf supervised learning strategies, including state-of-art deep learning models, and a selection of small-to-moderate-sized basic supervised learning benchmark data sets.
(ii) On the selection of benchmark data sets representative for basic (non-specialized) supervised learning, the best performing algorithms are ensembles of trees and kernel-based algorithms. Neural networks (deep or not) perform poorly in comparison.
(iii) Of the algorithms benchmarked, grid-tuned support vector classifiers are the most demanding of computation time. Neural networks (deep or not) and the other algorithms benchmarked require computation time in a comparable orders of magnitude.

5.6.2 Limitations

The main limitations of our study are:

(i) restriction to the Delgado data set collection. Our study is at most as representative for the methods' performance as the Delgado data set collection is for basic supervised learning.
(ii) training the neural networks for one epoch only. As described in 5.4.3 we believe we arrived at this choice following standard tuning protocol, but it requires further investigation, especially to rule out a mistake - or to corroborate evidence of a potential general issue of neural networks with basic supervised learning (i.e., not on image, audio, text data etc).
(iii) A relative small set of prediction strategies. While our study is an initial proof-of-concept for MLaut on commonly used algorithms, it did not include composite strategies (e.g., full pipelines), or the full selection available in state-of-art packages.

5.6.3 Comparison to the study of Delgado et al

In comparison to the benchmarking study of Fernández-Delgado et al. [19], for most algorithms we find comparable performances which are within 95% confidence bands (of ours). A notable major departure is performance of the neural networks, which we find to be substantially worse. The latter finding may be plausibly explained by at least one of the following:

(i) an issue-in-principle with how we tuned the neural networks - e.g., a mistake; or a difference to how the neural networks were tuned by Fernández-Delgado et al. [19]. However, it appears that Fernández-Delgado et al. [19] used default settings.
(ii) an overly optimistic bias of Fernández-Delgado et al. [19], through their mistake of tuning on the test set. This bias would be expected to be most severe for the models with most degrees of freedom to tune - i.e., the neural networks.

In additional comparison, the general rankings (when disregarding the neural networks) are similar. Though, since a replication of rankings is dependent on conducting the study on exactly the same set of strategies, we are only able to state this qualitatively. Conversely, our confidence intervals indicate that rankings in general are very unstable on the data set collection, as roughly a half of the 179 classifiers which Fernández-Delgado et al. [19] benchmarked seem to be within 95% confidence ranges of each other.

This seems to highlight the crucial necessity of reporting not only performances but also confidence bands, if reasoning is to be conducted about which algorithmic strategies are the "best" performing ones.

5.6.4 Conclusions

Our findings corroborate most of the findings of the major existing benchmarking study of Fernández-Delgado et al. [19]. In addition, we validate the usefulness of MLaut to easily conduct such a study.

As a notable exception to this confirmation of results, we find that neural networks do not perform well on “basic” supervised classification data sets. While it may be explained by a bias that Fernández-Delgado et al. [19] introduced into their study by the mistake of tuning on the test set, it is still under the strong caveat that further investigation needs to be carried out, in particular with respect to the tuning behaviour of said networks, and our experiment not containing other mistakes.

However, if further investigation confirms our findings, it would be consistent with the findings of one of the original dropout papers [42], in which the authors also conclude that the improvements are more noticeable on image datasets and less so on other types of data such as text. For example, the authors found that the performance improvements achieved on the Reuters RCV1 corpus were not significant in comparison with architectures that did not use dropout. Furthermore, at least in our study we found no evidence to suggest that deep architectures performed better than shallow ones. In fact the 12 layer deep neural network architecture ranked just slightly better than our baseline classifier. Our findings also may suggest that wide architectures tend to perform better than thin ones on our training data. It should also be pointed out that the datasets we used in this experiment were relatively small in size. Therefore, it could be argued that deep neural networks can easily overfit such data, the default parameter choices and standard procedures are not appropriate - especially since such common practice may arguably be strongly adapted to image/audio/text data.

In terms of training time, the SVC algorithm proved to be the most expensive, taking on average almost 30 min to train in our set-up. However, it should be noted that this is due to the relatively large hyper-parameter search space that we used. On the other hand, among the top five algorithms the Bagging Classifier was one of the least expensive ones to train taking an average of only 5 seconds. Our top performer, the Random Forest Classifier, was also relatively inexpensive to train taking an average of only 14 seconds.

As our main finding, however, we consider the ease with which a user may generate the above results, using MLaut. The reader may (hopefully) convince themselves of this by inspecting the code and jupyter notebooks in the repository . We are also very appreciative of any criticism, or suggestions for improvement, made (say, by an unconvinced reader) through the project’s issue tracker.

References

[1] scikit-learn laboratory. `https://skll.readthedocs.io`. URL `https://skll.readthedocs.io`.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[3] Lei Jimmy Ba and Rich Caruana. Do Deep Nets Really Need to be Deep? *arXiv:1312.6184 [cs]*, 2013.

[4] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533 [cs]*, 2012.

[5] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 2012.

[6] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL `http://www.jmlr.org/papers/v17/15-066.html`.

[7] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006. ISBN 78-1-4939-3843-8.

[8] Leo Breiman. Bagging predictors. *Machine Learning*, 1996.

[9] Leo Breiman. Random Forests. *Machine Learning*, 2001.

[10] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. Api design for machine learning software: experiences from the scikit-learn project. *CoRR*, 2013.

[11] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv:1512.01274 [cs]*, 2015.

[12] François Chollet. Keras, 2015. URL `https://keras.io`.

[13] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 1995.

[14] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 1967.

[15] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 2006.

[16] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 2006.

[17] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*, 2013.

[18] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence and Data Science*. Institute of Mathematical Statistics Monographs. Cambridge University Press, Cambridge, 2016.

[19] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, 2014.

[20] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[21] Jerome Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 2001.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[23] Tarun Kumar Gupta and Khalid Raza. Optimizing Deep Neural Network Architecture: A Tabu Search Based Approach. *arXiv:1808.05979 [cs, stat]*, 2018.

[24] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[25] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures. *arXiv:1608.06037 [cs]*, 2016.

[26] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, 2012.

[27] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification. page 16, 2003.

[28] Jin Huang, Jingjing Lu, and Charles Ling. Comparing naive Bayes, decision trees, and SVM with AUC and accuracy. IEEE Comput. Soc, 2003.

[29] Sudhir Jagtap and Bheemashankar Kodge. Census Data Mining and Data Analysis using WEKA. *arXiv:1310.4647 [cs]*, 2013.

[30] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *Introduction to Statistical Learning*. Springer Publishing Company, Incorporated, 2013. ISBN 978-1-4614-7137-0.

[31] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding, 2014.

[32] Viktor Kazakov and Franz Király. mlaut: Machine Learning automation toolbox, 2018. URL `https://github.com/alan-turing-institute/mlaut`.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 2017.

[34] Varun Kumar Ojha, Ajith Abraham, and Václav Snášel. Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research. *Engineering Applications of Artificial Intelligence*, 2017.

[35] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 2011.

[36] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *arXiv:1802.09596 [stat]*, 2018.

[37] Sheldon M Ross. *Introductory Statistics - 3rd Edition*. Academic Press, 2010.

[38] Emanuele Sansone and Francesco G. B. De Natale. Training Feedforward Neural Networks with Standard Logistic Activations is Feasible. *arXiv:1710.01013 [cs, stat]*, 2017.

[39] Scikit-Learn. Model selection: choosing estimators and their parameters — scikit-learn 0.20.0 documentation, 2018.

[40] Frank Seide and Amit Agarwal. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2135–2135, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2945397.

[41] Sören Sonnenburg, Gunnar Rätsch, Sebastian Henschel, Christian Widmer, Jonas Behr, Alexander Zien, Fabio de Bona, Alexander Binder, Christian Gehl, and Vojtěch Franc. The SHOGUN Machine Learning Toolbox. *Journal of Machine Learning Research*, pages 1799–1802, 2010.

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014.

[43] Maksim Terpilowski. scikit-posthocs: Statistical post-hoc analysis and outlier detection algorithms, 2018. URL `http://github.com/maximtrp/scikit-posthocs`.

[44] Janek Thomas, Stefan Coors, and Bernd Bischl. Automatic gradient boosting. *arXiv preprint arXiv:1807.03873*, 2018.

[45] Jacques Wainer. Comparison of 14 different families of classification algorithms on 115 binary datasets. *arXiv:1606.00930 [cs]*, 2016.

[46] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1945.

[47] Max Kuhn Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R. Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and and Tyler Hunt. caret: Classification and Regression Training, 2018.

A Further benchmarking results

A.1 paired t-test, without multiple testing correction

	BaggingClassifier		BaselineClassifier		BernoulliNaiveBayes		GaussianNaiveBayes	
	t_stat	p_val	t_stat	p_val	t_stat	p_val	t_stat	p_val
BaggingClassifier	0.000	1.000	16.779	0.000	5.456	0.000	6.137	0.000
BaselineClassifier	-16.779	0.000	0.000	1.000	-11.764	0.000	-9.373	0.000
BernoulliNaiveBayes	-5.456	0.000	11.764	0.000	0.000	1.000	1.366	0.173
GaussianNaiveBayes	-6.137	0.000	9.373	0.000	-1.366	0.173	0.000	1.000
GradientBoostingClassifier	-1.407	0.161	14.703	0.000	3.719	0.000	4.610	0.000
K_Neighbours	-0.734	0.463	16.373	0.000	4.837	0.000	5.600	0.000
NN-12-layer_wide_with_dropout	-10.631	0.000	3.963	0.000	-6.273	0.000	-4.629	0.000
NN-12-layer_wide_with_dropout_lr01	-13.058	0.000	2.045	0.042	-8.562	0.000	-6.687	0.000
NN-12-layer_wide_with_dropout_lr1	-13.606	0.000	1.309	0.192	-9.183	0.000	-7.296	0.000
NN-2-layer-droput-input-layer_lr001	-6.453	0.000	8.206	0.000	-2.001	0.047	-0.660	0.510
NN-2-layer-droput-input-layer_lr01	-10.186	0.000	4.101	0.000	-5.927	0.000	-4.347	0.000
NN-2-layer-droput-input-layer_lr1	-11.574	0.000	3.020	0.003	-7.230	0.000	-5.521	0.000
NN-4-layer-droput-each-layer_lr0001	-5.912	0.000	8.451	0.000	-1.557	0.121	-0.278	0.781
NN-4-layer-droput-each-layer_lr01	-12.754	0.000	2.109	0.036	-8.336	0.000	-6.514	0.000
NN-4-layer-droput-each-layer_lr1	-12.640	0.000	2.350	0.020	-8.177	0.000	-6.346	0.000
NN-4-layer_thin_dropout	-6.405	0.000	8.003	0.000	-2.042	0.042	-0.723	0.471
NN-4-layer_thin_dropout_lr01	-12.112	0.000	2.299	0.022	-7.839	0.000	-6.117	0.000
NN-4-layer_thin_dropout_lr1	-13.293	0.000	1.411	0.159	-8.937	0.000	-7.100	0.000
NN-4-layer_wide_no_dropout	-4.958	0.000	9.704	0.000	-0.477	0.634	0.742	0.459
NN-4-layer_wide_no_dropout_lr01	-12.554	0.000	2.500	0.013	-8.067	0.000	-6.234	0.000
NN-4-layer_wide_no_dropout_lr1	-12.618	0.000	2.316	0.021	-8.174	0.000	-6.351	0.000
NN-4-layer_wide_with_dropout	-5.043	0.000	9.661	0.000	-0.548	0.584	0.680	0.497
NN-4-layer_wide_with_dropout_lr01	-13.170	0.000	1.892	0.060	-8.690	0.000	-6.813	0.000
NN-4-layer_wide_with_dropout_lr1	-12.877	0.000	2.118	0.035	-8.416	0.000	-6.568	0.000
PassiveAggressiveClassifier	-2.876	0.004	13.497	0.000	2.313	0.022	3.371	0.001
RandomForestClassifier	0.253	0.800	16.752	0.000	5.597	0.000	6.262	0.000
SVC	-0.607	0.544	15.781	0.000	4.660	0.000	5.443	0.000

	GradientBoostingClassifier		K_Neighbours		NN-12-layer_wide_with_dropout		NN-12-layer_wide_with_dropout_lr01	
	t_stat	p_val	t_stat	p_val	t_stat	p_val	t_stat	p_val
BaggingClassifier	1.407	0.161	0.734	0.463	10.631	0.000	13.058	0.000
BaselineClassifier	-14.703	0.000	-16.373	0.000	-3.963	0.000	-2.045	0.042
BernoulliNaiveBayes	-3.719	0.000	-4.837	0.000	6.273	0.000	8.562	0.000
GaussianNaiveBayes	-4.610	0.000	-5.600	0.000	4.629	0.000	6.687	0.000
GradientBoostingClassifier	0.000	1.000	-0.749	0.454	9.091	0.000	11.374	0.000
K_Neighbours	0.749	0.454	0.000	1.000	10.190	0.000	12.634	0.000
NN-12-layer_wide_with_dropout	-9.091	0.000	-10.190	0.000	0.000	1.000	1.837	0.068
NN-12-layer_wide_with_dropout_lr01	-11.374	0.000	-12.634	0.000	-1.837	0.068	0.000	1.000
NN-12-layer_wide_with_dropout_lr1	-11.936	0.000	-13.193	0.000	-2.470	0.014	-0.665	0.507
NN-2-layer-droput-input-layer_lr001	-5.027	0.000	-5.953	0.000	3.805	0.000	5.751	0.000
NN-2-layer-droput-input-layer_lr01	-8.702	0.000	-9.748	0.000	0.190	0.850	2.002	0.046
NN-2-layer-droput-input-layer_lr1	-10.008	0.000	-11.144	0.000	-0.855	0.394	0.961	0.338
NN-4-layer-droput-each-layer_lr0001	-4.541	0.000	-5.415	0.000	4.099	0.000	6.021	0.000
NN-4-layer-droput-each-layer_lr01	-11.115	0.000	-12.332	0.000	-1.734	0.084	0.084	0.933
NN-4-layer-droput-each-layer_lr1	-10.985	0.000	-12.214	0.000	-1.540	0.125	0.295	0.768
NN-4-layer_thin_dropout	-5.013	0.000	-5.914	0.000	3.686	0.000	5.602	0.000
NN-4-layer_thin_dropout_lr01	-10.559	0.000	-11.693	0.000	-1.474	0.142	0.310	0.757
NN-4-layer_thin_dropout_lr1	-11.665	0.000	-12.881	0.000	-2.338	0.020	-0.549	0.584
NN-4-layer_wide_no_dropout	-3.581	0.000	-4.436	0.000	5.141	0.000	7.126	0.000
NN-4-layer_wide_no_dropout_lr01	-10.891	0.000	-12.125	0.000	-1.416	0.158	0.427	0.670
NN-4-layer_wide_no_dropout_lr1	-10.972	0.000	-12.193	0.000	-1.560	0.120	0.269	0.788
NN-4-layer_wide_with_dropout	-3.658	0.000	-4.520	0.000	5.091	0.000	7.079	0.000
NN-4-layer_wide_with_dropout_lr01	-11.489	0.000	-12.748	0.000	-1.968	0.050	-0.138	0.891
NN-4-layer_wide_with_dropout_lr1	-11.215	0.000	-12.454	0.000	-1.751	0.081	0.079	0.937
PassiveAggressiveClassifier	-1.370	0.172	-2.238	0.026	7.982	0.000	10.251	0.000
RandomForestClassifier	1.618	0.107	0.976	0.330	10.700	0.000	13.096	0.000
SVC	0.791	0.430	0.086	0.931	9.919	0.000	12.269	0.000

BaggingClassifier
BaselineClassifier
BernoulliNaiveBay
GaussianNaiveBay
GradientBoosting
K_Neighbours
NN-12-layer_wide_
NN-12-layer_wide_
NN-12-layer_wide_
NN-2-layer-droput
NN-2-layer-droput
NN-2-layer-droput
NN-4-layer-droput
NN-4-layer-droput
NN-4-layer-droput
NN-4-layer_thin_d
NN-4-layer_thin_d
NN-4-layer_thin_d
NN-4-layer_wide_n
NN-4-layer_wide_n
NN-4-layer_wide_n
NN-4-layer_wide_w
NN-4-layer_wide_w
NN-4-layer_wide_w
PassiveAggressive
RandomForestCla
SVC

	NN-4-layer-droput-each-layer_lr0001		NN-4-layer-droput-each-layer_lr01		NN-4-layer-droput-each-layer_lr1		NN-4-layer_thin_dropout	
	t_stat	p_val	t_stat	p_val	t_stat	p_val	t_stat	p_val
BaggingClassifier	5.912	0.000	12.754	0.000	12.640	0.000	6.405	0.000
BaselineClassifier	-8.451	0.000	-2.109	0.036	-2.350	0.020	-8.003	0.000
BernoulliNaiveBayes	1.557	0.121	8.336	0.000	8.177	0.000	2.042	0.042
GaussianNaiveBayes	0.278	0.781	6.514	0.000	6.346	0.000	0.723	0.471
GradientBoostingClassifier	4.541	0.000	11.115	0.000	10.985	0.000	5.013	0.000
K_Neighbours	5.415	0.000	12.332	0.000	12.214	0.000	5.914	0.000
NN-12-layer_wide_with_dropout	-4.099	0.000	1.734	0.084	1.540	0.125	-3.686	0.000
NN-12-layer_wide_with_dropout_lr01	-6.021	0.000	-0.084	0.933	-0.295	0.768	-5.602	0.000
NN-12-layer_wide_with_dropout_lr1	-6.609	0.000	-0.741	0.460	-0.954	0.341	-6.194	0.000
NN-2-layer-droput-input-layer_lr001	-0.354	0.724	5.600	0.000	5.430	0.000	0.070	0.945
NN-2-layer-droput-input-layer_lr01	-3.846	0.000	1.899	0.059	1.708	0.089	-3.439	0.001
NN-2-layer-droput-input-layer_lr1	-4.944	0.000	0.868	0.386	0.668	0.505	-4.533	0.000
NN-4-layer-droput-each-layer_lr0001	0.000	1.000	5.870	0.000	5.704	0.000	0.418	0.677
NN-4-layer-droput-each-layer_lr01	-5.870	0.000	0.000	1.000	-0.208	0.835	-5.455	0.000
NN-4-layer-droput-each-layer_lr1	-5.704	0.000	0.208	0.835	0.000	1.000	-5.286	0.000
NN-4-layer_thin_dropout	-0.418	0.677	5.455	0.000	5.286	0.000	0.000	1.000
NN-4-layer_thin_dropout_lr01	-5.518	0.000	0.225	0.822	0.023	0.982	-5.112	0.000
NN-4-layer_thin_dropout_lr1	-6.435	0.000	-0.625	0.533	-0.835	0.404	-6.024	0.000
NN-4-layer_wide_no_dropout	0.963	0.337	6.956	0.000	6.796	0.000	1.390	0.166
NN-4-layer_wide_no_dropout_lr01	-5.595	0.000	0.338	0.735	0.130	0.896	-5.175	0.000
NN-4-layer_wide_no_dropout_lr1	-5.712	0.000	0.183	0.855	-0.025	0.980	-5.296	0.000
NN-4-layer_wide_with_dropout	0.905	0.367	6.910	0.000	6.749	0.000	1.333	0.184
NN-4-layer_wide_with_dropout_lr01	-6.142	0.000	-0.220	0.826	-0.431	0.667	-5.724	0.000
NN-4-layer_wide_with_dropout_lr1	-5.914	0.000	-0.006	0.996	-0.215	0.830	-5.496	0.000
PassiveAggressiveClassifier	3.400	0.001	10.008	0.000	9.867	0.000	3.874	0.000
RandomForestClassifier	6.036	0.000	12.797	0.000	12.684	0.000	6.524	0.000
SVC	5.296	0.000	11.988	0.000	11.867	0.000	5.777	0.000

	NN-4-layer_thin_dropout_lr01		NN-4-layer_thin_dropout_lr1		NN-4-layer_wide_no_dropout		NN-4-layer_wide_no_dropout_lr01	
	t_stat	p_val	t_stat	p_val	t_stat	p_val	t_stat	p_val
BaggingClassifier	12.112	0.000	13.293	0.000	4.958	0.000	12.554	0.000
BaselineClassifier	-2.299	0.022	-1.411	0.159	-9.704	0.000	-2.500	0.013
BernoulliNaiveBayes	7.839	0.000	8.937	0.000	0.477	0.634	8.067	0.000
GaussianNaiveBayes	6.117	0.000	7.100	0.000	-0.742	0.459	6.234	0.000
GradientBoostingClassifier	10.559	0.000	11.665	0.000	3.581	0.000	10.891	0.000
K_Neighbours	11.693	0.000	12.881	0.000	4.436	0.000	12.125	0.000
NN-12-layer_wide_with_dropout	1.474	0.142	2.338	0.020	-5.141	0.000	1.416	0.158
NN-12-layer_wide_with_dropout_lr01	-0.310	0.757	0.549	0.584	-7.126	0.000	-0.427	0.670
NN-12-layer_wide_with_dropout_lr1	-0.950	0.343	-0.109	0.914	-7.710	0.000	-1.087	0.278
NN-2-layer-droput-input-layer_lr001	5.247	0.000	6.174	0.000	-1.339	0.182	5.319	0.000
NN-2-layer-droput-input-layer_lr01	1.640	0.102	2.493	0.013	-4.864	0.000	1.587	0.114
NN-2-layer-droput-input-layer_lr1	0.627	0.531	1.479	0.141	-5.999	0.000	0.542	0.588
NN-4-layer-droput-each-layer_lr0001	5.518	0.000	6.435	0.000	-0.963	0.337	5.595	0.000
NN-4-layer-droput-each-layer_lr01	-0.225	0.822	0.625	0.533	-6.956	0.000	-0.338	0.735
NN-4-layer-droput-each-layer_lr1	-0.023	0.982	0.835	0.404	-6.796	0.000	-0.130	0.896
NN-4-layer_thin_dropout	5.112	0.000	6.024	0.000	-1.390	0.166	5.175	0.000
NN-4-layer_thin_dropout_lr01	0.000	1.000	0.835	0.405	-6.569	0.000	-0.104	0.917
NN-4-layer_thin_dropout_lr1	-0.835	0.405	0.000	1.000	-7.519	0.000	-0.966	0.335
NN-4-layer_wide_no_dropout	6.569	0.000	7.519	0.000	0.000	1.000	6.690	0.000
NN-4-layer_wide_no_dropout_lr01	0.104	0.917	0.966	0.335	-6.690	0.000	0.000	1.000
NN-4-layer_wide_no_dropout_lr1	-0.047	0.963	0.809	0.420	-6.801	0.000	-0.155	0.877
NN-4-layer_wide_with_dropout	6.523	0.000	7.474	0.000	-0.061	0.951	6.642	0.000
NN-4-layer_wide_with_dropout_lr01	-0.442	0.659	0.413	0.680	-7.246	0.000	-0.564	0.574
NN-4-layer_wide_with_dropout_lr1	-0.232	0.817	0.623	0.534	-7.010	0.000	-0.346	0.729
PassiveAggressiveClassifier	9.480	0.000	10.574	0.000	2.402	0.017	9.767	0.000
RandomForestClassifier	12.165	0.000	13.332	0.000	5.096	0.000	12.598	0.000
SVC	11.391	0.000	12.532	0.000	4.342	0.000	11.777	0.000

BaggingClassifier
BaselineClassifier
BernoulliNaiveBa
GaussianNaiveBa
GradientBoosting
K_Neighbours
NN-12-layer_wide
NN-12-layer_wide
NN-12-layer_wide
NN-2-layer-dropu
NN-2-layer-dropu
NN-2-layer-dropu
NN-4-layer-dropu
NN-4-layer-dropu
NN-4-layer-dropu
NN-4-layer_thin_
NN-4-layer_thin_
NN-4-layer_thin_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
PassiveAggressiv
RandomForestCl
SVC

	PassiveAggressiveClassifier		RandomForestClassifier		SVC	
	t_stat	p_val	t_stat	p_val	t_stat	p_val
BaggingClassifier	2.876	0.004	-0.253	0.800	0.607	0.544
BaselineClassifier	-13.497	0.000	-16.752	0.000	-15.781	0.000
BernoulliNaiveBayes	-2.313	0.022	-5.597	0.000	-4.660	0.000
GaussianNaiveBayes	-3.371	0.001	-6.262	0.000	-5.443	0.000
GradientBoostingClassifier	1.370	0.172	-1.618	0.107	-0.791	0.430
K_Neighbours	2.238	0.026	-0.976	0.330	-0.086	0.931
NN-12-layer_wide_with_dropout	-7.982	0.000	-10.700	0.000	-9.919	0.000
NN-12-layer_wide_with_dropout_lr01	-10.251	0.000	-13.096	0.000	-12.269	0.000
NN-12-layer_wide_with_dropout_lr1	-10.834	0.000	-13.640	0.000	-12.823	0.000
NN-2-layer-droput-input-layer_lr001	-3.867	0.000	-6.571	0.000	-5.808	0.000
NN-2-layer-droput-input-layer_lr01	-7.614	0.000	-10.261	0.000	-9.503	0.000
NN-2-layer-droput-input-layer_lr1	-8.909	0.000	-11.632	0.000	-10.848	0.000
NN-4-layer-droput-each-layer_lr0001	-3.400	0.001	-6.036	0.000	-5.296	0.000
NN-4-layer-droput-each-layer_lr01	-10.008	0.000	-12.797	0.000	-11.988	0.000
NN-4-layer-droput-each-layer_lr1	-9.867	0.000	-12.684	0.000	-11.867	0.000
NN-4-layer_thin_dropout	-3.874	0.000	-6.524	0.000	-5.777	0.000
NN-4-layer_thin_dropout_lr01	-9.480	0.000	-12.165	0.000	-11.391	0.000
NN-4-layer_thin_dropout_lr1	-10.574	0.000	-13.332	0.000	-12.532	0.000
NN-4-layer_wide_no_dropout	-2.402	0.017	-5.096	0.000	-4.342	0.000
NN-4-layer_wide_no_dropout_lr01	-9.767	0.000	-12.598	0.000	-11.777	0.000
NN-4-layer_wide_no_dropout_lr1	-9.858	0.000	-12.663	0.000	-11.849	0.000
NN-4-layer_wide_with_dropout	-2.476	0.014	-5.180	0.000	-4.422	0.000
NN-4-layer_wide_with_dropout_lr01	-10.371	0.000	-13.207	0.000	-12.382	0.000
NN-4-layer_wide_with_dropout_lr1	-10.098	0.000	-12.918	0.000	-12.099	0.000
PassiveAggressiveClassifier	0.000	1.000	-3.062	0.002	-2.205	0.028
RandomForestClassifier	3.062	0.002	0.000	1.000	0.839	0.402
SVC	2.205	0.028	-0.839	0.402	0.000	1.000

A.2 Wilcoxon signed-rank test, without Bonferroni correction

	BaggingClassifier		BaselineClassifier		BernoulliNaiveBayes		GaussianNaiveBayes	
	statistic	p_val	statistic	p_val	statistic	p_val	statistic	p_val
BaggingClassifier	0.0	NaN	6.0	0.0	443.5	0.000	411.5	0.000
BaselineClassifier	6.0	0.000	0.0	NaN	49.0	0.000	400.0	0.000
BernoulliNaiveBayes	443.5	0.000	49.0	0.0	0.0	NaN	2228.0	0.056
GaussianNaiveBayes	411.5	0.000	400.0	0.0	2228.0	0.056	0.0	NaN
GradientBoostingClassifier	984.0	0.000	23.5	0.0	1073.5	0.000	1053.0	0.000
K_Neighbours	1929.0	0.020	1.0	0.0	306.0	0.000	344.0	0.000
NN-12-layer_wide_with_dropout	218.0	0.000	574.0	0.0	822.5	0.000	1587.0	0.000
NN-12-layer_wide_with_dropout_lr01	148.0	0.000	1061.0	0.0	362.0	0.000	984.0	0.000
NN-12-layer_wide_with_dropout_lr1	119.0	0.000	1393.0	0.0	331.0	0.000	755.0	0.000
NN-2-layer-droput-input-layer_lr001	475.0	0.000	52.0	0.0	2260.5	0.036	2944.5	0.873
NN-2-layer-droput-input-layer_lr01	142.0	0.000	239.0	0.0	410.5	0.000	1259.0	0.000
NN-2-layer-droput-input-layer_lr1	174.0	0.000	649.0	0.0	417.5	0.000	1191.0	0.000
NN-4-layer-droput-each-layer_lr0001	633.0	0.000	53.0	0.0	2665.5	0.316	2394.0	0.214
NN-4-layer-droput-each-layer_lr01	122.0	0.000	1023.0	0.0	349.0	0.000	992.0	0.000
NN-4-layer-droput-each-layer_lr1	139.0	0.000	732.0	0.0	363.0	0.000	1034.0	0.000
NN-4-layer_thin_dropout	507.0	0.000	60.0	0.0	2357.0	0.053	2788.5	0.755
NN-4-layer_thin_dropout_lr01	118.0	0.000	1118.0	0.0	338.5	0.000	944.5	0.000
NN-4-layer_thin_dropout_lr1	103.0	0.000	1555.0	0.0	325.0	0.000	947.0	0.000
NN-4-layer_wide_no_dropout	702.5	0.000	27.0	0.0	2550.5	0.369	1891.5	0.007
NN-4-layer_wide_no_dropout_lr01	106.0	0.000	816.0	0.0	337.0	0.000	1002.0	0.000
NN-4-layer_wide_no_dropout_lr1	112.0	0.000	899.0	0.0	348.0	0.000	1007.0	0.000
NN-4-layer_wide_with_dropout	698.5	0.000	40.0	0.0	2860.0	0.466	2144.0	0.014
NN-4-layer_wide_with_dropout_lr01	123.0	0.000	1212.5	0.0	333.0	0.000	984.0	0.000
NN-4-layer_wide_with_dropout_lr1	114.0	0.000	999.5	0.0	302.0	0.000	988.0	0.000
PassiveAggressiveClassifier	980.0	0.000	11.5	0.0	1330.0	0.000	958.5	0.000
RandomForestClassifier	1459.0	0.005	0.0	0.0	259.5	0.000	252.5	0.000
SVC	2571.0	0.606	0.0	0.0	387.5	0.000	249.0	0.000

	GradientBoostingClassifier		K_Neighbours		NN-12-layer_wide_with_dropout		NN-12-layer_wide_with_dropout_lr01	
	statistic	p_val	statistic	p_val	statistic	p_val	statistic	p_val
BaggingClassifier	984.0	0.000	1929.0	0.020	218.0	0.000	148.0	0.000
BaselineClassifier	23.5	0.000	1.0	0.000	574.0	0.000	1061.0	0.000
BernoulliNaiveBayes	1073.5	0.000	306.0	0.000	822.5	0.000	362.0	0.000
GaussianNaiveBayes	1053.0	0.000	344.0	0.000	1587.0	0.000	984.0	0.000
GradientBoostingClassifier	0.0	NaN	2202.5	0.118	320.0	0.000	241.0	0.000
K_Neighbours	2202.5	0.118	0.0	NaN	121.0	0.000	51.0	0.000
NN-12-layer_wide_with_dropout	320.0	0.000	121.0	0.000	0.0	NaN	274.0	0.000
NN-12-layer_wide_with_dropout_lr01	241.0	0.000	51.0	0.000	274.0	0.000	0.0	NaN
NN-12-layer_wide_with_dropout_lr1	221.0	0.000	23.0	0.000	214.0	0.000	541.0	0.808
NN-2-layer-droput-input-layer_lr001	846.5	0.000	230.0	0.000	744.0	0.000	301.0	0.000
NN-2-layer-droput-input-layer_lr01	312.0	0.000	35.0	0.000	1269.5	0.656	463.5	0.000
NN-2-layer-droput-input-layer_lr1	285.0	0.000	66.0	0.000	681.0	0.025	451.0	0.005
NN-4-layer-droput-each-layer_lr0001	946.5	0.000	380.0	0.000	330.0	0.000	87.0	0.000
NN-4-layer-droput-each-layer_lr01	229.0	0.000	24.0	0.000	269.5	0.000	398.0	0.872
NN-4-layer-droput-each-layer_lr1	230.0	0.000	39.0	0.000	294.0	0.000	232.0	0.386
NN-4-layer_thin_dropout	756.0	0.000	361.0	0.000	637.0	0.000	310.0	0.000
NN-4-layer_thin_dropout_lr01	205.0	0.000	24.0	0.000	415.0	0.001	657.0	0.250
NN-4-layer_thin_dropout_lr1	222.0	0.000	25.0	0.000	194.5	0.000	394.0	0.239
NN-4-layer_wide_no_dropout	1107.5	0.000	562.0	0.000	380.5	0.000	172.5	0.000
NN-4-layer_wide_no_dropout_lr01	222.0	0.000	23.0	0.000	370.0	0.001	353.5	0.149
NN-4-layer_wide_no_dropout_lr1	220.0	0.000	26.0	0.000	420.5	0.000	326.0	0.372
NN-4-layer_wide_with_dropout	1109.0	0.000	529.0	0.000	361.5	0.000	169.0	0.000
NN-4-layer_wide_with_dropout_lr01	211.0	0.000	41.0	0.000	294.5	0.000	389.0	0.778
NN-4-layer_wide_with_dropout_lr1	223.0	0.000	26.0	0.000	290.0	0.000	375.0	0.472
PassiveAggressiveClassifier	2379.5	0.062	1178.0	0.000	435.0	0.000	229.0	0.000
RandomForestClassifier	732.0	0.000	1187.0	0.000	109.0	0.000	82.0	0.000
SVC	1797.5	0.001	1867.5	0.067	58.0	0.000	39.0	0.000

	NN-12-layer_wide_with_dropout_lr1		NN-2-layer-droput-input-layer_lr001		NN-2-layer-droput-input-layer_lr01		NN-2-layer-droput-input-layer_lr1	
	statistic	p_val	statistic	p_val	statistic	p_val	statistic	p_val
BaggingClassifier	119.0	0.000	475.0	0.000	142.0	0.000	174.0	0.000
BaselineClassifier	1393.0	0.000	52.0	0.000	239.0	0.000	649.0	0.000
BernoulliNaiveBayes	331.0	0.000	2260.5	0.036	410.5	0.000	417.5	0.000
GaussianNaiveBayes	755.0	0.000	2944.5	0.873	1259.0	0.000	1191.0	0.000
GradientBoostingClassifier	221.0	0.000	846.5	0.000	312.0	0.000	285.0	0.000
K_Neighbours	23.0	0.000	230.0	0.000	35.0	0.000	66.0	0.000
NN-12-layer_wide_with_dropout	214.0	0.000	744.0	0.000	1269.5	0.656	681.0	0.025
NN-12-layer_wide_with_dropout_lr01	541.0	0.808	301.0	0.000	463.5	0.000	451.0	0.005
NN-12-layer_wide_with_dropout_lr1	0.0	NaN	203.0	0.000	399.0	0.000	468.0	0.002
NN-2-layer-droput-input-layer_lr001	203.0	0.000	0.0	NaN	442.0	0.000	365.0	0.000
NN-2-layer-droput-input-layer_lr01	399.0	0.000	442.0	0.000	0.0	NaN	703.0	0.010
NN-2-layer-droput-input-layer_lr1	468.0	0.002	365.0	0.000	703.0	0.010	0.0	NaN
NN-4-layer-droput-each-layer_lr0001	91.0	0.000	1645.5	0.106	170.0	0.000	138.0	0.000
NN-4-layer-droput-each-layer_lr01	536.0	0.767	238.0	0.000	360.0	0.000	436.0	0.003
NN-4-layer-droput-each-layer_lr1	332.0	0.202	364.0	0.000	522.5	0.000	485.0	0.017
NN-4-layer_thin_dropout	310.0	0.000	2456.5	0.814	517.0	0.000	434.0	0.000
NN-4-layer_thin_dropout_lr01	545.5	0.060	382.5	0.000	626.5	0.001	739.0	0.138
NN-4-layer_thin_dropout_lr1	489.5	0.430	227.0	0.000	383.0	0.000	478.5	0.001
NN-4-layer_wide_no_dropout	165.0	0.000	1309.0	0.000	160.0	0.000	206.5	0.000
NN-4-layer_wide_no_dropout_lr01	331.0	0.056	253.0	0.000	575.5	0.001	675.5	0.163
NN-4-layer_wide_no_dropout_lr1	403.5	0.198	289.0	0.000	544.0	0.001	462.5	0.025
NN-4-layer_wide_with_dropout	189.0	0.000	1468.5	0.000	238.0	0.000	223.5	0.000
NN-4-layer_wide_with_dropout_lr01	490.0	0.756	264.0	0.000	424.0	0.000	397.0	0.002
NN-4-layer_wide_with_dropout_lr1	506.0	0.289	225.0	0.000	451.5	0.000	610.0	0.010
PassiveAggressiveClassifier	173.0	0.000	776.0	0.000	200.0	0.000	288.0	0.000
RandomForestClassifier	61.0	0.000	210.0	0.000	73.0	0.000	104.0	0.000
SVC	19.0	0.000	254.0	0.000	40.0	0.000	68.0	0.000

	NN-4-layer-droput-each-layer_lr0001		NN-4-layer-droput-each-layer_lr01		NN-4-layer-droput-each-layer_lr1		NN-4-layer_thin_dropout	
	statistic	p_val	statistic	p_val	statistic	p_val	statistic	p_val
BaggingClassifier	633.0	0.000	122.0	0.000	139.0	0.000	507.0	0.000
BaselineClassifier	53.0	0.000	1023.0	0.000	732.0	0.000	60.0	0.000
BernoulliNaiveBayes	2665.5	0.316	349.0	0.000	363.0	0.000	2357.0	0.053
GaussianNaiveBayes	2394.0	0.214	992.0	0.000	1034.0	0.000	2788.5	0.755
GradientBoostingClassifier	946.5	0.000	229.0	0.000	230.0	0.000	756.0	0.000
K_Neighbours	380.0	0.000	24.0	0.000	39.0	0.000	361.0	0.000
NN-12-layer_wide_with_dropout	330.0	0.000	269.5	0.000	294.0	0.000	637.0	0.000
NN-12-layer_wide_with_dropout_lr01	87.0	0.000	398.0	0.872	232.0	0.386	310.0	0.000
NN-12-layer_wide_with_dropout_lr1	91.0	0.000	536.0	0.767	332.0	0.202	310.0	0.000
NN-2-layer-droput-input-layer_lr001	1645.5	0.106	238.0	0.000	364.0	0.000	2456.5	0.814
NN-2-layer-droput-input-layer_lr01	170.0	0.000	360.0	0.000	522.5	0.000	517.0	0.000
NN-2-layer-droput-input-layer_lr1	138.0	0.000	436.0	0.003	485.0	0.017	434.0	0.000
NN-4-layer-droput-each-layer_lr0001	0.0	NaN	89.0	0.000	135.0	0.000	1816.0	0.085
NN-4-layer-droput-each-layer_lr01	89.0	0.000	0.0	NaN	411.0	0.613	316.5	0.000
NN-4-layer-droput-each-layer_lr1	135.0	0.000	411.0	0.613	0.0	NaN	366.0	0.000
NN-4-layer_thin_dropout	1816.0	0.085	316.5	0.000	366.0	0.000	0.0	NaN
NN-4-layer_thin_dropout_lr01	200.0	0.000	510.0	0.218	703.0	0.734	425.5	0.000
NN-4-layer_thin_dropout_lr1	96.0	0.000	338.0	0.103	250.0	0.019	301.5	0.000
NN-4-layer_wide_no_dropout	1216.5	0.000	152.0	0.000	183.0	0.000	1345.5	0.000
NN-4-layer_wide_no_dropout_lr01	78.0	0.000	305.0	0.158	395.0	0.346	320.0	0.000
NN-4-layer_wide_no_dropout_lr1	130.0	0.000	331.0	0.567	379.5	0.682	362.5	0.000
NN-4-layer_wide_with_dropout	1222.0	0.000	180.0	0.000	170.0	0.000	1367.0	0.000
NN-4-layer_wide_with_dropout_lr01	103.0	0.000	306.5	0.497	305.0	0.236	298.0	0.000
NN-4-layer_wide_with_dropout_lr1	77.0	0.000	331.0	0.757	358.0	0.856	237.0	0.000
PassiveAggressiveClassifier	1205.0	0.000	207.0	0.000	208.0	0.000	1111.0	0.000
RandomForestClassifier	296.5	0.000	64.0	0.000	73.0	0.000	198.0	0.000
SVC	321.5	0.000	23.0	0.000	29.0	0.000	197.0	0.000

	NN-4-layer_thin_dropout_lr01 statistic	p_val	NN-4-layer_thin_dropout_lr1 statistic	p_val	NN-4-layer_wide_no_dropout statistic	p_val	NN-4-layer_wide_no_dropout_lr01 statistic	p_val
BaggingClassifier	118.0	0.000	103.0	0.000	702.5	0.000	106.0	0.000
BaselineClassifier	1118.0	0.000	1555.0	0.000	27.0	0.000	816.0	0.000
BernoulliNaiveBayes	338.5	0.000	325.0	0.000	2550.5	0.369	337.0	0.000
GaussianNaiveBayes	944.5	0.000	947.0	0.000	1891.5	0.007	1002.0	0.000
GradientBoostingClassifier	205.0	0.000	222.0	0.000	1107.5	0.000	222.0	0.000
K_Neighbours	24.0	0.000	25.0	0.000	562.0	0.000	23.0	0.000
NN-12-layer_wide_with_dropout	415.0	0.001	194.5	0.000	380.5	0.000	370.0	0.001
NN-12-layer_wide_with_dropout_lr01	657.0	0.250	394.0	0.239	172.5	0.000	353.5	0.149
NN-12-layer_wide_with_dropout_lr1	545.5	0.060	489.5	0.430	165.0	0.000	331.0	0.056
NN-2-layer-droput-input-layer_lr001	382.5	0.000	227.0	0.000	1309.0	0.000	253.0	0.000
NN-2-layer-droput-input-layer_lr01	626.5	0.001	383.0	0.000	160.0	0.000	575.5	0.001
NN-2-layer-droput-input-layer_lr1	739.0	0.138	478.5	0.001	206.5	0.000	675.5	0.163
NN-4-layer-droput-each-layer_lr0001	200.0	0.000	96.0	0.000	1216.5	0.000	78.0	0.000
NN-4-layer-droput-each-layer_lr01	510.0	0.218	338.0	0.103	152.0	0.000	305.0	0.158
NN-4-layer-droput-each-layer_lr1	703.0	0.734	250.0	0.019	183.0	0.000	395.0	0.346
NN-4-layer_thin_dropout	425.5	0.000	301.5	0.000	1345.5	0.000	320.0	0.000
NN-4-layer_thin_dropout_lr01	0.0	NaN	468.0	0.028	193.0	0.000	824.5	0.987
NN-4-layer_thin_dropout_lr1	468.0	0.028	0.0	NaN	129.0	0.000	226.0	0.005
NN-4-layer_wide_no_dropout	193.0	0.000	129.0	0.000	0.0	NaN	154.0	0.000
NN-4-layer_wide_no_dropout_lr01	824.5	0.987	226.0	0.005	154.0	0.000	0.0	NaN
NN-4-layer_wide_no_dropout_lr1	658.0	0.611	253.0	0.056	154.0	0.000	442.5	0.540
NN-4-layer_wide_with_dropout	221.5	0.000	115.0	0.000	2055.0	0.503	141.0	0.000
NN-4-layer_wide_with_dropout_lr01	559.0	0.166	464.0	0.403	165.0	0.000	366.0	0.132
NN-4-layer_wide_with_dropout_lr1	645.5	0.297	526.0	0.093	135.0	0.000	530.0	0.552
PassiveAggressiveClassifier	191.0	0.000	160.0	0.000	1667.0	0.000	170.0	0.000
RandomForestClassifier	72.0	0.000	59.0	0.000	359.0	0.000	59.0	0.000
SVC	24.0	0.000	23.0	0.000	422.0	0.000	22.0	0.000

BaggingClassifier
BaselineClassifier
BernoulliNaiveBa
GaussianNaiveBa
GradientBoosting
K_Neighbours
NN-12-layer_wide
NN-12-layer_wide
NN-12-layer_wide
NN-2-layer-dropu
NN-2-layer-dropu
NN-2-layer-dropu
NN-4-layer-dropu
NN-4-layer-dropu
NN-4-layer-dropu
NN-4-layer_thin_
NN-4-layer_thin_
NN-4-layer_thin_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
PassiveAggressiv
RandomForestCl
SVC

	PassiveAggressiveClassifier		RandomForestClassifier		SVC	
	statistic	p_val	statistic	p_val	statistic	p_val
BaggingClassifier	980.0	0.000	1459.0	0.005	2571.0	0.606
BaselineClassifier	11.5	0.000	0.0	0.000	0.0	0.000
BernoulliNaiveBayes	1330.0	0.000	259.5	0.000	387.5	0.000
GaussianNaiveBayes	958.5	0.000	252.5	0.000	249.0	0.000
GradientBoostingClassifier	2379.5	0.062	732.0	0.000	1797.5	0.001
K_Neighbours	1178.0	0.000	1187.0	0.000	1867.5	0.067
NN-12-layer_wide_with_dropout	435.0	0.000	109.0	0.000	58.0	0.000
NN-12-layer_wide_with_dropout_lr01	229.0	0.000	82.0	0.000	39.0	0.000
NN-12-layer_wide_with_dropout_lr1	173.0	0.000	61.0	0.000	19.0	0.000
NN-2-layer-droput-input-layer_lr001	776.0	0.000	210.0	0.000	254.0	0.000
NN-2-layer-droput-input-layer_lr01	200.0	0.000	73.0	0.000	40.0	0.000
NN-2-layer-droput-input-layer_lr1	288.0	0.000	104.0	0.000	68.0	0.000
NN-4-layer-droput-each-layer_lr0001	1205.0	0.000	296.5	0.000	321.5	0.000
NN-4-layer-droput-each-layer_lr01	207.0	0.000	64.0	0.000	23.0	0.000
NN-4-layer-droput-each-layer_lr1	208.0	0.000	73.0	0.000	29.0	0.000
NN-4-layer_thin_dropout	1111.0	0.000	198.0	0.000	197.0	0.000
NN-4-layer_thin_dropout_lr01	191.0	0.000	72.0	0.000	24.0	0.000
NN-4-layer_thin_dropout_lr1	160.0	0.000	59.0	0.000	23.0	0.000
NN-4-layer_wide_no_dropout	1667.0	0.000	359.0	0.000	422.0	0.000
NN-4-layer_wide_no_dropout_lr01	170.0	0.000	59.0	0.000	22.0	0.000
NN-4-layer_wide_no_dropout_lr1	174.0	0.000	64.0	0.000	23.0	0.000
NN-4-layer_wide_with_dropout	1705.5	0.000	412.0	0.000	501.5	0.000
NN-4-layer_wide_with_dropout_lr01	198.0	0.000	77.0	0.000	28.0	0.000
NN-4-layer_wide_with_dropout_lr1	164.0	0.000	69.0	0.000	23.0	0.000
PassiveAggressiveClassifier	0.0	NaN	668.0	0.000	601.5	0.000
RandomForestClassifier	668.0	0.000	0.0	NaN	1828.5	0.008
SVC	601.5	0.000	1828.5	0.008	0.0	NaN

A.3 Neményi post-hoc significance

	BaggingClassifier	BaselineClassifier	BernoulliNaiveBayes	GaussianNaiveBayes
BaggingClassifier	-1.000	0.000	0.938	0.597
BaselineClassifier	0.000	-1.000	0.000	0.000
BernoulliNaiveBayes	0.938	0.000	-1.000	1.000
GaussianNaiveBayes	0.597	0.000	1.000	-1.000
GradientBoostingClassifier	1.000	0.000	1.000	0.973
K_Neighbours	1.000	0.000	0.988	0.822
NN-12-layer_wide_with_dropout	0.000	0.986	0.509	0.903
NN-12-layer_wide_with_dropout_lr01	0.000	1.000	0.010	0.121
NN-12-layer_wide_with_dropout_lr1	0.000	1.000	0.002	0.036
NN-2-layer-droput-input-layer_lr001	0.338	0.002	1.000	1.000
NN-2-layer-droput-input-layer_lr01	0.000	0.969	0.624	0.947
NN-2-layer-droput-input-layer_lr1	0.000	1.000	0.149	0.567
NN-4-layer-droput-each-layer_lr0001	0.574	0.000	1.000	1.000
NN-4-layer-droput-each-layer_lr01	0.000	1.000	0.016	0.163
NN-4-layer-droput-each-layer_lr1	0.000	1.000	0.025	0.212
NN-4-layer_thin_dropout	0.308	0.002	1.000	1.000
NN-4-layer_thin_dropout_lr01	0.000	1.000	0.048	0.314
NN-4-layer_thin_dropout_lr1	0.000	1.000	0.003	0.054
NN-4-layer_wide_no_dropout	0.946	0.000	1.000	1.000
NN-4-layer_wide_no_dropout_lr01	0.000	1.000	0.032	0.245
NN-4-layer_wide_no_dropout_lr1	0.000	1.000	0.024	0.209
NN-4-layer_wide_with_dropout	0.942	0.000	1.000	1.000
NN-4-layer_wide_with_dropout_lr01	0.000	1.000	0.008	0.100
NN-4-layer_wide_with_dropout_lr1	0.000	1.000	0.015	0.158
PassiveAggressiveClassifier	1.000	0.000	1.000	1.000
RandomForestClassifier	1.000	0.000	0.871	0.447
SVC	1.000	0.000	0.977	0.749

	GradientBoostingClassifier	K_Neighbours	NN-12-layer_wide_with_dropout	NN-12-layer_wide_with_dropout_lr01
BaggingClassifier	1.000	1.000	0.000	0.000
BaselineClassifier	0.000	0.000	0.986	1.000
BernoulliNaiveBayes	1.000	0.988	0.509	0.010
GaussianNaiveBayes	0.973	0.822	0.903	0.121
GradientBoostingClassifier	-1.000	1.000	0.000	0.000
K_Neighbours	1.000	-1.000	0.000	0.000
NN-12-layer_wide_with_dropout	0.000	0.000	-1.000	1.000
NN-12-layer_wide_with_dropout_lr01	0.000	0.000	1.000	-1.000
NN-12-layer_wide_with_dropout_lr1	0.000	0.000	1.000	1.000
NN-2-layer-droput-input-layer_lr001	0.885	0.594	0.979	0.299
NN-2-layer-droput-input-layer_lr01	0.000	0.000	1.000	1.000
NN-2-layer-droput-input-layer_lr1	0.000	0.000	1.000	1.000
NN-4-layer-droput-each-layer_lr0001	0.969	0.806	0.914	0.133
NN-4-layer-droput-each-layer_lr01	0.000	0.000	1.000	1.000
NN-4-layer-droput-each-layer_lr1	0.000	0.000	1.000	1.000
NN-4-layer_thin_dropout	0.867	0.561	0.983	0.329
NN-4-layer_thin_dropout_lr01	0.000	0.000	1.000	1.000
NN-4-layer_thin_dropout_lr1	0.000	0.000	1.000	1.000
NN-4-layer_wide_no_dropout	1.000	0.990	0.486	0.009
NN-4-layer_wide_no_dropout_lr01	0.000	0.000	1.000	1.000
NN-4-layer_wide_no_dropout_lr1	0.000	0.000	1.000	1.000
NN-4-layer_wide_with_dropout	1.000	0.990	0.496	0.009
NN-4-layer_wide_with_dropout_lr01	0.000	0.000	1.000	1.000
NN-4-layer_wide_with_dropout_lr1	0.000	0.000	1.000	1.000
PassiveAggressiveClassifier	1.000	1.000	0.006	0.000
RandomForestClassifier	1.000	1.000	0.000	0.000
SVC	1.000	1.000	0.000	0.000

BaggingClassifier
BaselineClassifier
BernoulliNaiveBaye
GaussianNaiveBaye
GradientBoostingC
K_Neighbours
NN-12-layer_wide_w
NN-12-layer_wide_w
NN-12-layer_wide_w
NN-2-layer-droput-i
NN-2-layer-droput-i
NN-2-layer-droput-i
NN-4-layer-droput-e
NN-4-layer-droput-e
NN-4-layer-droput-e
NN-4-layer_thin_dro
NN-4-layer_thin_dro
NN-4-layer_thin_dro
NN-4-layer_wide_no
NN-4-layer_wide_no
NN-4-layer_wide_no
NN-4-layer_wide_wi
NN-4-layer_wide_wi
NN-4-layer_wide_wi
PassiveAggressiveC
RandomForestClass
SVC

	NN-4-layer-droput-each-layer_lr0001	NN-4-layer-droput-each-layer_lr01	NN-4-layer-droput-each-layer_lr1	NN-4-layer_thin_dropout
BaggingClassifier	0.574	0.000	0.000	0.308
BaselineClassifier	0.000	1.000	1.000	0.002
BernoulliNaiveBayes	1.000	0.016	0.025	1.000
GaussianNaiveBayes	1.000	0.163	0.212	1.000
GradientBoostingClassifier	0.969	0.000	0.000	0.867
K_Neighbours	0.806	0.000	0.000	0.561
NN-12-layer_wide_with_dropout	0.914	1.000	1.000	0.983
NN-12-layer_wide_with_dropout_lr01	0.133	1.000	1.000	0.329
NN-12-layer_wide_with_dropout_lr1	0.041	1.000	1.000	0.139
NN-2-layer-droput-input-layer_lr001	1.000	0.369	0.441	1.000
NN-2-layer-droput-input-layer_lr01	0.953	1.000	1.000	0.993
NN-2-layer-droput-input-layer_lr1	0.590	1.000	1.000	0.827
NN-4-layer-droput-each-layer_lr0001	-1.000	0.177	0.229	1.000
NN-4-layer-droput-each-layer_lr01	0.177	-1.000	1.000	0.401
NN-4-layer-droput-each-layer_lr1	0.229	1.000	-1.000	0.475
NN-4-layer_thin_dropout	1.000	0.401	0.475	-1.000
NN-4-layer_thin_dropout_lr01	0.335	1.000	1.000	0.604
NN-4-layer_thin_dropout_lr1	0.060	1.000	1.000	0.185
NN-4-layer_wide_no_dropout	1.000	0.014	0.022	1.000
NN-4-layer_wide_no_dropout_lr01	0.264	1.000	1.000	0.521
NN-4-layer_wide_no_dropout_lr1	0.225	1.000	1.000	0.470
NN-4-layer_wide_with_dropout	1.000	0.015	0.023	1.000
NN-4-layer_wide_with_dropout_lr01	0.110	1.000	1.000	0.287
NN-4-layer_wide_with_dropout_lr1	0.172	1.000	1.000	0.392
PassiveAggressiveClassifier	1.000	0.000	0.000	0.996
RandomForestClassifier	0.424	0.000	0.000	0.193
SVC	0.730	0.000	0.000	0.464

	NN-4-layer_thin_dropout_lr01	NN-4-layer_thin_dropout_lr1	NN-4-layer_wide_no_dropout	NN-4-layer_wide_no_dropout_lr01
BaggingClassifier	0.000	0.000	0.946	0.000
BaselineClassifier	1.000	1.000	0.000	1.000
BernoulliNaiveBayes	0.048	0.003	1.000	0.032
GaussianNaiveBayes	0.314	0.054	1.000	0.245
GradientBoostingClassifier	0.000	0.000	1.000	0.000
K_Neighbours	0.000	0.000	0.990	0.000
NN-12-layer_wide_with_dropout	1.000	1.000	0.486	1.000
NN-12-layer_wide_with_dropout_lr01	1.000	1.000	0.009	1.000
NN-12-layer_wide_with_dropout_lr1	1.000	1.000	0.001	1.000
NN-2-layer-droput-input-layer_lr001	0.570	0.164	1.000	0.487
NN-2-layer-droput-input-layer_lr01	1.000	1.000	0.601	1.000
NN-2-layer-droput-input-layer_lr1	1.000	1.000	0.136	1.000
NN-4-layer-droput-each-layer_lr0001	0.335	0.060	1.000	0.264
NN-4-layer-droput-each-layer_lr01	1.000	1.000	0.014	1.000
NN-4-layer-droput-each-layer_lr1	1.000	1.000	0.022	1.000
NN-4-layer_thin_dropout	0.604	0.185	1.000	0.521
NN-4-layer_thin_dropout_lr01	-1.000	1.000	0.043	1.000
NN-4-layer_thin_dropout_lr1	1.000	-1.000	0.003	1.000
NN-4-layer_wide_no_dropout	0.043	0.003	-1.000	0.028
NN-4-layer_wide_no_dropout_lr01	1.000	1.000	0.028	-1.000
NN-4-layer_wide_no_dropout_lr1	1.000	1.000	0.021	1.000
NN-4-layer_wide_with_dropout	0.045	0.003	1.000	0.029
NN-4-layer_wide_with_dropout_lr01	1.000	1.000	0.007	1.000
NN-4-layer_wide_with_dropout_lr1	1.000	1.000	0.013	1.000
PassiveAggressiveClassifier	0.000	0.000	1.000	0.000
RandomForestClassifier	0.000	0.000	0.884	0.000
SVC	0.000	0.000	0.981	0.000

BaggingClassifier
BaselineClassifier
BernoulliNaiveBa
GaussianNaiveBa
GradientBoostin
K_Neighbours
NN-12-layer_wid
NN-12-layer_wid
NN-12-layer_wid
NN-2-layer-dropu
NN-2-layer-dropu
NN-2-layer-dropu
NN-4-layer-dropu
NN-4-layer-dropu
NN-4-layer-dropu
NN-4-layer_thin_
NN-4-layer_thin_
NN-4-layer_thin_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
NN-4-layer_wide_
PassiveAggressiv
RandomForestCl
SVC

	PassiveAggressiveClassifier	RandomForestClassifier	SVC
BaggingClassifier	1.000	1.000	1.000
BaselineClassifier	0.000	0.000	0.000
BernoulliNaiveBayes	1.000	0.871	0.977
GaussianNaiveBayes	1.000	0.447	0.749
GradientBoostingClassifier	1.000	1.000	1.000
K_Neighbours	1.000	1.000	1.000
NN-12-layer_wide_with_dropout	0.006	0.000	0.000
NN-12-layer_wide_with_dropout_lr01	0.000	0.000	0.000
NN-12-layer_wide_with_dropout_lr1	0.000	0.000	0.000
NN-2-layer-droput-input-layer_lr001	0.997	0.216	0.498
NN-2-layer-droput-input-layer_lr01	0.011	0.000	0.000
NN-2-layer-droput-input-layer_lr1	0.000	0.000	0.000
NN-4-layer-droput-each-layer_lr0001	1.000	0.424	0.730
NN-4-layer-droput-each-layer_lr01	0.000	0.000	0.000
NN-4-layer-droput-each-layer_lr1	0.000	0.000	0.000
NN-4-layer_thin_dropout	0.996	0.193	0.464
NN-4-layer_thin_dropout_lr01	0.000	0.000	0.000
NN-4-layer_thin_dropout_lr1	0.000	0.000	0.000
NN-4-layer_wide_no_dropout	1.000	0.884	0.981
NN-4-layer_wide_no_dropout_lr01	0.000	0.000	0.000
NN-4-layer_wide_no_dropout_lr1	0.000	0.000	0.000
NN-4-layer_wide_with_dropout	1.000	0.878	0.979
NN-4-layer_wide_with_dropout_lr01	0.000	0.000	0.000
NN-4-layer_wide_with_dropout_lr1	0.000	0.000	0.000
PassiveAggressiveClassifier	-1.000	1.000	1.000
RandomForestClassifier	1.000	-1.000	1.000
SVC	1.000	1.000	-1.000

A.4 Accuracy by data set, type (a) standard errors

		loss	std_error
abalone	BaggingClassifier	0.37708	0.01305
	BaselineClassifier	0.66715	0.01269
	BernoulliNaiveBayes	0.44888	0.01339
	GaussianNaiveBayes	0.44017	0.01337
	GradientBoostingClassifier	0.38869	0.01313
	K_Neighbours	0.36476	0.01296
	NN-12-layer_wide_with_dropout	0.36766	0.01298
	NN-12-layer_wide_with_dropout_lr01	0.65627	0.01279
	NN-12-layer_wide_with_dropout_lr1	0.65627	0.01279
	NN-2-layer-droput-input-layer_lr001	0.62872	0.01301
	NN-2-layer-droput-input-layer_lr01	0.47208	0.01344
	NN-2-layer-droput-input-layer_lr1	0.58231	0.01328
	NN-4-layer-droput-each-layer_lr0001	0.35025	0.01285
	NN-4-layer-droput-each-layer_lr01	0.67368	0.01263
	NN-4-layer-droput-each-layer_lr1	0.67368	0.01263
	NN-4-layer_thin_dropout	0.38724	0.01312
	NN-4-layer_thin_dropout_lr01	0.65627	0.01279
	NN-4-layer_thin_dropout_lr1	0.65627	0.01279
	NN-4-layer_wide_no_dropout	0.36186	0.01294
	NN-4-layer_wide_no_dropout_lr01	0.57360	0.01332
	NN-4-layer_wide_no_dropout_lr1	0.67005	0.01266
	NN-4-layer_wide_with_dropout	0.37563	0.01304
	NN-4-layer_wide_with_dropout_lr01	0.67368	0.01263
	NN-4-layer_wide_with_dropout_lr1	0.67368	0.01263
	PassiveAggressiveClassifier	0.37346	0.01303
	RandomForestClassifier	0.36331	0.01295
	SVC	0.35460	0.01288
acute_inflammation	BaggingClassifier	0.00000	0.00000
	BaselineClassifier	0.32500	0.07406
	BernoulliNaiveBayes	0.12500	0.05229
	GaussianNaiveBayes	0.17500	0.06008
	GradientBoostingClassifier	0.00000	0.00000
	K_Neighbours	0.00000	0.00000
	NN-12-layer_wide_with_dropout	0.47500	0.07896
	NN-12-layer_wide_with_dropout_lr01	0.52500	0.07896
	NN-12-layer_wide_with_dropout_lr1	0.52500	0.07896
	NN-2-layer-droput-input-layer_lr001	0.00000	0.00000
	NN-2-layer-droput-input-layer_lr01	0.15000	0.05646
	NN-2-layer-droput-input-layer_lr1	0.35000	0.07542
	NN-4-layer-droput-each-layer_lr0001	0.00000	0.00000
	NN-4-layer-droput-each-layer_lr01	0.52500	0.07896
	NN-4-layer-droput-each-layer_lr1	0.47500	0.07896
	NN-4-layer_thin_dropout	0.05000	0.03446
	NN-4-layer_thin_dropout_lr01	0.45000	0.07866
	NN-4-layer_thin_dropout_lr1	0.52500	0.07896
	NN-4-layer_wide_no_dropout	0.00000	0.00000
	NN-4-layer_wide_no_dropout_lr01	0.52500	0.07896
	NN-4-layer_wide_no_dropout_lr1	0.52500	0.07896
	NN-4-layer_wide_with_dropout	0.00000	0.00000
	NN-4-layer_wide_with_dropout_lr01	0.52500	0.07896
	NN-4-layer_wide_with_dropout_lr1	0.47500	0.07896
	PassiveAggressiveClassifier	0.00000	0.00000

Continued on next page

		loss	std_error
	RandomForestClassifier	0.00000	0.00000
	SVC	0.20000	0.06325
acute_nephritis	BaggingClassifier	0.00000	0.00000
	BaselineClassifier	0.57500	0.07816
	BernoulliNaiveBayes	0.02500	0.02469
	GaussianNaiveBayes	0.02500	0.02469
	GradientBoostingClassifier	0.00000	0.00000
	K_Neighbours	0.00000	0.00000
	NN-12-layer_wide_with_dropout	0.50000	0.07906
	NN-12-layer_wide_with_dropout_lr01	0.50000	0.07906
	NN-12-layer_wide_with_dropout_lr1	0.50000	0.07906
	NN-2-layer-droput-input-layer_lr001	0.00000	0.00000
	NN-2-layer-droput-input-layer_lr01	0.00000	0.00000
	NN-2-layer-droput-input-layer_lr1	0.50000	0.07906
	NN-4-layer-droput-each-layer_lr0001	0.00000	0.00000
	NN-4-layer-droput-each-layer_lr01	0.50000	0.07906
	NN-4-layer-droput-each-layer_lr1	0.50000	0.07906
	NN-4-layer_thin_dropout	0.17500	0.06008
	NN-4-layer_thin_dropout_lr01	0.50000	0.07906
	NN-4-layer_thin_dropout_lr1	0.50000	0.07906
	NN-4-layer_wide_no_dropout	0.00000	0.00000
	NN-4-layer_wide_no_dropout_lr01	0.50000	0.07906
	NN-4-layer_wide_no_dropout_lr1	0.50000	0.07906
	NN-4-layer_wide_with_dropout	0.00000	0.00000
	NN-4-layer_wide_with_dropout_lr01	0.50000	0.07906
	NN-4-layer_wide_with_dropout_lr1	0.50000	0.07906
	PassiveAggressiveClassifier	0.00000	0.00000
	RandomForestClassifier	0.00000	0.00000
	SVC	0.00000	0.00000
adult	BaggingClassifier	0.14772	0.00279
	BaselineClassifier	0.36158	0.00378
	BernoulliNaiveBayes	0.19612	0.00313
	GaussianNaiveBayes	0.19121	0.00310
	GradientBoostingClassifier	0.13314	0.00268
	K_Neighbours	0.16286	0.00291
	NN-12-layer_wide_with_dropout	0.23855	0.00336
	NN-12-layer_wide_with_dropout_lr01	0.23855	0.00336
	NN-12-layer_wide_with_dropout_lr1	0.23855	0.00336
	NN-2-layer-droput-input-layer_lr001	0.16894	0.00295
	NN-2-layer-droput-input-layer_lr01	0.23855	0.00336
	NN-2-layer-droput-input-layer_lr1	0.23855	0.00336
	NN-4-layer-droput-each-layer_lr0001	0.15523	0.00285
	NN-4-layer-droput-each-layer_lr01	0.23855	0.00336
	NN-4-layer-droput-each-layer_lr1	0.23855	0.00336
	NN-4-layer_thin_dropout	0.15188	0.00283
	NN-4-layer_thin_dropout_lr01	0.23855	0.00336
	NN-4-layer_thin_dropout_lr1	0.23855	0.00336
	NN-4-layer_wide_no_dropout	0.15231	0.00283
	NN-4-layer_wide_no_dropout_lr01	0.23855	0.00336
	NN-4-layer_wide_no_dropout_lr1	0.23855	0.00336
	NN-4-layer_wide_with_dropout	0.15238	0.00283
	NN-4-layer_wide_with_dropout_lr01	0.23855	0.00336
	NN-4-layer_wide_with_dropout_lr1	0.23855	0.00336
	PassiveAggressiveClassifier	0.17856	0.00302
	RandomForestClassifier	0.14617	0.00278

Continued on next page

		loss	std_error
	SVC	0.15213	0.00283
annealing	BaggingClassifier	0.04714	0.01230
	BaselineClassifier	0.42424	0.02868
	BernoulliNaiveBayes	0.15825	0.02118
	GaussianNaiveBayes	0.47138	0.02897
	GradientBoostingClassifier	0.05051	0.01271
	K_Neighbours	0.12795	0.01938
	NN-12-layer_wide_with_dropout	0.24579	0.02498
	NN-12-layer_wide_with_dropout_lr01	0.24579	0.02498
	NN-12-layer_wide_with_dropout_lr1	0.99327	0.00475
	NN-2-layer-droput-input-layer_lr001	0.24579	0.02498
	NN-2-layer-droput-input-layer_lr01	0.24579	0.02498
	NN-2-layer-droput-input-layer_lr1	0.24579	0.02498
	NN-4-layer-droput-each-layer_lr0001	0.24579	0.02498
	NN-4-layer-droput-each-layer_lr01	0.24579	0.02498
	NN-4-layer-droput-each-layer_lr1	0.24579	0.02498
	NN-4-layer_thin_dropout	0.24579	0.02498
	NN-4-layer_thin_dropout_lr01	0.24579	0.02498
	NN-4-layer_thin_dropout_lr1	0.24579	0.02498
	NN-4-layer_wide_no_dropout	0.24579	0.02498
	NN-4-layer_wide_no_dropout_lr01	0.24579	0.02498
	NN-4-layer_wide_no_dropout_lr1	0.24579	0.02498
	NN-4-layer_wide_with_dropout	0.24579	0.02498
	NN-4-layer_wide_with_dropout_lr01	0.24579	0.02498
	NN-4-layer_wide_with_dropout_lr1	0.24579	0.02498
	PassiveAggressiveClassifier	0.19192	0.02285
	RandomForestClassifier	0.05724	0.01348
	SVC	0.10438	0.01774
arrhythmia	BaggingClassifier	0.25333	0.03551
	BaselineClassifier	0.68667	0.03787
	BernoulliNaiveBayes	0.42000	0.04030
	GaussianNaiveBayes	0.84667	0.02942
	GradientBoostingClassifier	0.29333	0.03717
	K_Neighbours	0.43333	0.04046
	NN-12-layer_wide_with_dropout	0.48000	0.04079
	NN-12-layer_wide_with_dropout_lr01	0.48000	0.04079
	NN-12-layer_wide_with_dropout_lr1	0.48000	0.04079
	NN-2-layer-droput-input-layer_lr001	0.42667	0.04038
	NN-2-layer-droput-input-layer_lr01	0.48000	0.04079
	NN-2-layer-droput-input-layer_lr1	0.78000	0.03382
	NN-4-layer-droput-each-layer_lr0001	0.48000	0.04079
	NN-4-layer-droput-each-layer_lr01	0.48000	0.04079
	NN-4-layer-droput-each-layer_lr1	0.48000	0.04079
	NN-4-layer_thin_dropout	0.48000	0.04079
	NN-4-layer_thin_dropout_lr01	0.95333	0.01722
	NN-4-layer_thin_dropout_lr1	0.48000	0.04079
	NN-4-layer_wide_no_dropout	0.56667	0.04046
	NN-4-layer_wide_no_dropout_lr01	0.48000	0.04079
	NN-4-layer_wide_no_dropout_lr1	0.48000	0.04079
	NN-4-layer_wide_with_dropout	0.48000	0.04079
	NN-4-layer_wide_with_dropout_lr01	0.48000	0.04079
	NN-4-layer_wide_with_dropout_lr1	0.48000	0.04079
	PassiveAggressiveClassifier	0.36667	0.03935
	RandomForestClassifier	0.28667	0.03692
	SVC	0.26667	0.03611

Continued on next page

		loss	std_error
audiology_std	BaggingClassifier	0.27692	0.05550
	BaselineClassifier	0.86154	0.04284
	BernoulliNaiveBayes	0.35385	0.05931
	GaussianNaiveBayes	0.50769	0.06201
	GradientBoostingClassifier	0.30769	0.05725
	K_Neighbours	0.30769	0.05725
	NN-12-layer_wide_with_dropout	0.76923	0.05226
	NN-12-layer_wide_with_dropout_lr01	0.75385	0.05343
	NN-12-layer_wide_with_dropout_lr1	0.76923	0.05226
	NN-2-layer-droput-input-layer_lr001	0.81538	0.04812
	NN-2-layer-droput-input-layer_lr01	0.78462	0.05099
	NN-2-layer-droput-input-layer_lr1	0.75385	0.05343
	NN-4-layer-droput-each-layer_lr0001	0.75385	0.05343
	NN-4-layer-droput-each-layer_lr01	0.75385	0.05343
	NN-4-layer-droput-each-layer_lr1	0.75385	0.05343
	NN-4-layer_thin_dropout	0.95385	0.02602
	NN-4-layer_thin_dropout_lr01	0.75385	0.05343
	NN-4-layer_thin_dropout_lr1	0.75385	0.05343
	NN-4-layer_wide_no_dropout	0.75385	0.05343
	NN-4-layer_wide_no_dropout_lr01	0.75385	0.05343
	NN-4-layer_wide_no_dropout_lr1	0.75385	0.05343
	NN-4-layer_wide_with_dropout	0.89231	0.03845
	NN-4-layer_wide_with_dropout_lr01	0.75385	0.05343
	NN-4-layer_wide_with_dropout_lr1	0.75385	0.05343
	PassiveAggressiveClassifier	0.32308	0.05801
	RandomForestClassifier	0.81538	0.04812
	SVC	0.86154	0.04284
balance_scale	BaggingClassifier	0.20773	0.02820
	BaselineClassifier	0.62319	0.03368
	BernoulliNaiveBayes	0.31884	0.03239
	GaussianNaiveBayes	0.10628	0.02142
	GradientBoostingClassifier	0.18841	0.02718
	K_Neighbours	0.11594	0.02225
	NN-12-layer_wide_with_dropout	0.56039	0.03450
	NN-12-layer_wide_with_dropout_lr01	0.56039	0.03450
	NN-12-layer_wide_with_dropout_lr1	0.56039	0.03450
	NN-2-layer-droput-input-layer_lr001	0.14493	0.02447
	NN-2-layer-droput-input-layer_lr01	0.37681	0.03368
	NN-2-layer-droput-input-layer_lr1	0.42995	0.03441
	NN-4-layer-droput-each-layer_lr0001	0.11594	0.02225
	NN-4-layer-droput-each-layer_lr01	0.56039	0.03450
	NN-4-layer-droput-each-layer_lr1	0.56039	0.03450
	NN-4-layer_thin_dropout	0.19807	0.02770
	NN-4-layer_thin_dropout_lr01	0.40097	0.03406
	NN-4-layer_thin_dropout_lr1	0.56039	0.03450
	NN-4-layer_wide_no_dropout	0.09662	0.02053
	NN-4-layer_wide_no_dropout_lr01	0.52657	0.03470
	NN-4-layer_wide_no_dropout_lr1	0.56039	0.03450
	NN-4-layer_wide_with_dropout	0.09662	0.02053
	NN-4-layer_wide_with_dropout_lr01	0.52657	0.03470
	NN-4-layer_wide_with_dropout_lr1	0.56039	0.03450
	PassiveAggressiveClassifier	0.13043	0.02341
	RandomForestClassifier	0.14010	0.02412
	SVC	0.07729	0.01856
bank	BaggingClassifier	0.11394	0.00823

Continued on next page

		loss	std_error
	BaselineClassifier	0.22453	0.01080
	BernoulliNaiveBayes	0.15080	0.00926
	GaussianNaiveBayes	0.19437	0.01024
	GradientBoostingClassifier	0.10992	0.00810
	K_Neighbours	0.12131	0.00845
	NN-12-layer_wide_with_dropout	0.12399	0.00853
	NN-12-layer_wide_with_dropout_lr01	0.12399	0.00853
	NN-12-layer_wide_with_dropout_lr1	0.12399	0.00853
	NN-2-layer-droput-input-layer_lr001	0.12131	0.00845
	NN-2-layer-droput-input-layer_lr01	0.12399	0.00853
	NN-2-layer-droput-input-layer_lr1	0.12399	0.00853
	NN-4-layer-droput-each-layer_lr0001	0.12399	0.00853
	NN-4-layer-droput-each-layer_lr01	0.12399	0.00853
	NN-4-layer-droput-each-layer_lr1	0.12399	0.00853
	NN-4-layer_thin_dropout	0.10992	0.00810
	NN-4-layer_thin_dropout_lr01	0.12399	0.00853
	NN-4-layer_thin_dropout_lr1	0.12399	0.00853
	NN-4-layer_wide_no_dropout	0.12399	0.00853
	NN-4-layer_wide_no_dropout_lr01	0.12399	0.00853
	NN-4-layer_wide_no_dropout_lr1	0.12399	0.00853
	NN-4-layer_wide_with_dropout	0.12399	0.00853
	NN-4-layer_wide_with_dropout_lr01	0.12399	0.00853
	NN-4-layer_wide_with_dropout_lr1	0.12399	0.00853
	PassiveAggressiveClassifier	0.12466	0.00855
	RandomForestClassifier	0.11796	0.00835
	SVC	0.11595	0.00829
blood	BaggingClassifier	0.25911	0.02788
	BaselineClassifier	0.35628	0.03047
	BernoulliNaiveBayes	0.24696	0.02744
	GaussianNaiveBayes	0.25101	0.02759
	GradientBoostingClassifier	0.26316	0.02802
	K_Neighbours	0.20243	0.02557
	NN-12-layer_wide_with_dropout	0.25506	0.02774
	NN-12-layer_wide_with_dropout_lr01	0.25506	0.02774
	NN-12-layer_wide_with_dropout_lr1	0.25506	0.02774
	NN-2-layer-droput-input-layer_lr001	0.25506	0.02774
	NN-2-layer-droput-input-layer_lr01	0.25506	0.02774
	NN-2-layer-droput-input-layer_lr1	0.25506	0.02774
	NN-4-layer-droput-each-layer_lr0001	0.25506	0.02774
	NN-4-layer-droput-each-layer_lr01	0.25506	0.02774
	NN-4-layer-droput-each-layer_lr1	0.25506	0.02774
	NN-4-layer_thin_dropout	0.25506	0.02774
	NN-4-layer_thin_dropout_lr01	0.25506	0.02774
	NN-4-layer_thin_dropout_lr1	0.25506	0.02774
	NN-4-layer_wide_no_dropout	0.23482	0.02697
	NN-4-layer_wide_no_dropout_lr01	0.25506	0.02774
	NN-4-layer_wide_no_dropout_lr1	0.25506	0.02774
	NN-4-layer_wide_with_dropout	0.21862	0.02630
	NN-4-layer_wide_with_dropout_lr01	0.25506	0.02774
	NN-4-layer_wide_with_dropout_lr1	0.25506	0.02774
	PassiveAggressiveClassifier	0.24291	0.02729
	RandomForestClassifier	0.21053	0.02594
	SVC	0.25101	0.02759
breast_cancer	BaggingClassifier	0.25263	0.04458
	BaselineClassifier	0.43158	0.05082

Continued on next page

		loss	std_error
	BernoulliNaiveBayes	0.27368	0.04574
	GaussianNaiveBayes	0.31579	0.04769
	GradientBoostingClassifier	0.33684	0.04849
	K_Neighbours	0.25263	0.04458
	NN-12-layer_wide_with_dropout	0.29474	0.04678
	NN-12-layer_wide_with_dropout_lr01	0.29474	0.04678
	NN-12-layer_wide_with_dropout_lr1	0.29474	0.04678
	NN-2-layer-droput-input-layer_lr001	0.29474	0.04678
	NN-2-layer-droput-input-layer_lr01	0.29474	0.04678
	NN-2-layer-droput-input-layer_lr1	0.29474	0.04678
	NN-4-layer-droput-each-layer_lr0001	0.29474	0.04678
	NN-4-layer-droput-each-layer_lr01	0.29474	0.04678
	NN-4-layer-droput-each-layer_lr1	0.29474	0.04678
	NN-4-layer_thin_dropout	0.31579	0.04769
	NN-4-layer_thin_dropout_lr01	0.29474	0.04678
	NN-4-layer_thin_dropout_lr1	0.29474	0.04678
	NN-4-layer_wide_no_dropout	0.29474	0.04678
	NN-4-layer_wide_no_dropout_lr01	0.29474	0.04678
	NN-4-layer_wide_no_dropout_lr1	0.29474	0.04678
	NN-4-layer_wide_with_dropout	0.28421	0.04628
	NN-4-layer_wide_with_dropout_lr01	0.29474	0.04678
	NN-4-layer_wide_with_dropout_lr1	0.29474	0.04678
	PassiveAggressiveClassifier	0.29474	0.04678
	RandomForestClassifier	0.29474	0.04678
	SVC	0.28421	0.04628
breast_cancer_wisc	BaggingClassifier	0.04329	0.01339
	BaselineClassifier	0.51082	0.03289
	BernoulliNaiveBayes	0.02597	0.01047
	GaussianNaiveBayes	0.03463	0.01203
	GradientBoostingClassifier	0.06494	0.01621
	K_Neighbours	0.03896	0.01273
	NN-12-layer_wide_with_dropout	0.66667	0.03102
	NN-12-layer_wide_with_dropout_lr01	0.66667	0.03102
	NN-12-layer_wide_with_dropout_lr1	0.33333	0.03102
	NN-2-layer-droput-input-layer_lr001	0.03030	0.01128
	NN-2-layer-droput-input-layer_lr01	0.02597	0.01047
	NN-2-layer-droput-input-layer_lr1	0.33333	0.03102
	NN-4-layer-droput-each-layer_lr0001	0.03030	0.01128
	NN-4-layer-droput-each-layer_lr01	0.66667	0.03102
	NN-4-layer-droput-each-layer_lr1	0.33333	0.03102
	NN-4-layer_thin_dropout	0.03030	0.01128
	NN-4-layer_thin_dropout_lr01	0.13853	0.02273
	NN-4-layer_thin_dropout_lr1	0.66667	0.03102
	NN-4-layer_wide_no_dropout	0.01732	0.00858
	NN-4-layer_wide_no_dropout_lr01	0.66667	0.03102
	NN-4-layer_wide_no_dropout_lr1	0.66234	0.03112
	NN-4-layer_wide_with_dropout	0.05195	0.01460
	NN-4-layer_wide_with_dropout_lr01	0.66667	0.03102
	NN-4-layer_wide_with_dropout_lr1	0.31169	0.03048
	PassiveAggressiveClassifier	0.02165	0.00957
	RandomForestClassifier	0.03896	0.01273
	SVC	0.03030	0.01128
breast_cancer_wisc_diag	BaggingClassifier	0.06383	0.01783
	BaselineClassifier	0.48404	0.03645
	BernoulliNaiveBayes	0.09043	0.02092

Continued on next page

		loss	std_error
	GaussianNaiveBayes	0.06383	0.01783
	GradientBoostingClassifier	0.05851	0.01712
	K_Neighbours	0.04255	0.01472
	NN-12-layer_wide_with_dropout	0.34574	0.03469
	NN-12-layer_wide_with_dropout_lr01	0.34574	0.03469
	NN-12-layer_wide_with_dropout_lr1	0.34574	0.03469
	NN-2-layer-droput-input-layer_lr001	0.04787	0.01557
	NN-2-layer-droput-input-layer_lr01	0.10638	0.02249
	NN-2-layer-droput-input-layer_lr1	0.21277	0.02985
	NN-4-layer-droput-each-layer_lr0001	0.04787	0.01557
	NN-4-layer-droput-each-layer_lr01	0.65426	0.03469
	NN-4-layer-droput-each-layer_lr1	0.34574	0.03469
	NN-4-layer_thin_dropout	0.04787	0.01557
	NN-4-layer_thin_dropout_lr01	0.65426	0.03469
	NN-4-layer_thin_dropout_lr1	0.34574	0.03469
	NN-4-layer_wide_no_dropout	0.03723	0.01381
	NN-4-layer_wide_no_dropout_lr01	0.34574	0.03469
	NN-4-layer_wide_no_dropout_lr1	0.34574	0.03469
	NN-4-layer_wide_with_dropout	0.05319	0.01637
	NN-4-layer_wide_with_dropout_lr01	0.34574	0.03469
	NN-4-layer_wide_with_dropout_lr1	0.34574	0.03469
	PassiveAggressiveClassifier	0.03191	0.01282
	RandomForestClassifier	0.05319	0.01637
	SVC	0.04255	0.01472
breast_tissue	BaggingClassifier	0.28571	0.07636
	BaselineClassifier	0.82857	0.06370
	BernoulliNaiveBayes	0.40000	0.08281
	GaussianNaiveBayes	0.34286	0.08023
	GradientBoostingClassifier	0.37143	0.08167
	K_Neighbours	0.37143	0.08167
	NN-12-layer_wide_with_dropout	0.71429	0.07636
	NN-12-layer_wide_with_dropout_lr01	0.71429	0.07636
	NN-12-layer_wide_with_dropout_lr1	0.71429	0.07636
	NN-2-layer-droput-input-layer_lr001	0.62857	0.08167
	NN-2-layer-droput-input-layer_lr01	0.51429	0.08448
	NN-2-layer-droput-input-layer_lr1	0.88571	0.05378
	NN-4-layer-droput-each-layer_lr0001	0.40000	0.08281
	NN-4-layer-droput-each-layer_lr01	0.85714	0.05915
	NN-4-layer-droput-each-layer_lr1	0.71429	0.07636
	NN-4-layer_thin_dropout	0.51429	0.08448
	NN-4-layer_thin_dropout_lr01	0.88571	0.05378
	NN-4-layer_thin_dropout_lr1	0.88571	0.05378
	NN-4-layer_wide_no_dropout	0.48571	0.08448
	NN-4-layer_wide_no_dropout_lr01	0.71429	0.07636
	NN-4-layer_wide_no_dropout_lr1	0.88571	0.05378
	NN-4-layer_wide_with_dropout	0.48571	0.08448
	NN-4-layer_wide_with_dropout_lr01	0.88571	0.05378
	NN-4-layer_wide_with_dropout_lr1	0.71429	0.07636
	PassiveAggressiveClassifier	0.31429	0.07847
	RandomForestClassifier	0.22857	0.07098
	SVC	0.31429	0.07847
car	BaggingClassifier	0.03678	0.00788
	BaselineClassifier	0.47110	0.02089
	BernoulliNaiveBayes	0.27320	0.01865
	GaussianNaiveBayes	0.29072	0.01900

Continued on next page

		loss	std_error
	GradientBoostingClassifier	0.02977	0.00711
	K_Neighbours	0.06130	0.01004
	NN-12-layer_wide_with_dropout	0.14186	0.01460
	NN-12-layer_wide_with_dropout_lr01	0.29422	0.01907
	NN-12-layer_wide_with_dropout_lr1	0.29422	0.01907
	NN-2-layer-droput-input-layer_lr001	0.26270	0.01842
	NN-2-layer-droput-input-layer_lr01	0.29422	0.01907
	NN-2-layer-droput-input-layer_lr1	0.29422	0.01907
	NN-4-layer-droput-each-layer_lr0001	0.16287	0.01545
	NN-4-layer-droput-each-layer_lr01	0.29422	0.01907
	NN-4-layer-droput-each-layer_lr1	0.29422	0.01907
	NN-4-layer_thin_dropout	0.19790	0.01667
	NN-4-layer_thin_dropout_lr01	0.29422	0.01907
	NN-4-layer_thin_dropout_lr1	0.29422	0.01907
	NN-4-layer_wide_no_dropout	0.12609	0.01389
	NN-4-layer_wide_no_dropout_lr01	0.29422	0.01907
	NN-4-layer_wide_no_dropout_lr1	0.29422	0.01907
	NN-4-layer_wide_with_dropout	0.14186	0.01460
	NN-4-layer_wide_with_dropout_lr01	0.29422	0.01907
	NN-4-layer_wide_with_dropout_lr1	0.29422	0.01907
	PassiveAggressiveClassifier	0.20665	0.01694
	RandomForestClassifier	0.05429	0.00948
	SVC	0.09807	0.01245
cardiotocography_10clases	BaggingClassifier	0.14103	0.01314
	BaselineClassifier	0.84900	0.01351
	BernoulliNaiveBayes	0.35185	0.01802
	GaussianNaiveBayes	0.48575	0.01886
	GradientBoostingClassifier	0.15100	0.01351
	K_Neighbours	0.24501	0.01623
	NN-12-layer_wide_with_dropout	0.71652	0.01701
	NN-12-layer_wide_with_dropout_lr01	0.95014	0.00821
	NN-12-layer_wide_with_dropout_lr1	0.71652	0.01701
	NN-2-layer-droput-input-layer_lr001	0.34330	0.01792
	NN-2-layer-droput-input-layer_lr01	0.72222	0.01691
	NN-2-layer-droput-input-layer_lr1	0.71652	0.01701
	NN-4-layer-droput-each-layer_lr0001	0.32479	0.01767
	NN-4-layer-droput-each-layer_lr01	0.82194	0.01444
	NN-4-layer-droput-each-layer_lr1	0.84900	0.01351
	NN-4-layer_thin_dropout	0.41026	0.01856
	NN-4-layer_thin_dropout_lr01	0.71652	0.01701
	NN-4-layer_thin_dropout_lr1	0.84900	0.01351
	NN-4-layer_wide_no_dropout	0.29915	0.01728
	NN-4-layer_wide_no_dropout_lr01	0.71652	0.01701
	NN-4-layer_wide_no_dropout_lr1	0.88889	0.01186
	NN-4-layer_wide_with_dropout	0.33903	0.01787
	NN-4-layer_wide_with_dropout_lr01	0.84900	0.01351
	NN-4-layer_wide_with_dropout_lr1	0.84900	0.01351
	PassiveAggressiveClassifier	0.22080	0.01566
	RandomForestClassifier	0.14957	0.01346
	SVC	0.17521	0.01435
cardiotocography_3clases	BaggingClassifier	0.06268	0.00915
	BaselineClassifier	0.38034	0.01832
	BernoulliNaiveBayes	0.20655	0.01528
	GaussianNaiveBayes	0.29487	0.01721
	GradientBoostingClassifier	0.05128	0.00832

Continued on next page

		loss	std_error
	K_Neighbours	0.09544	0.01109
	NN-12-layer_wide_with_dropout	0.23362	0.01597
	NN-12-layer_wide_with_dropout_lr01	0.23362	0.01597
	NN-12-layer_wide_with_dropout_lr1	0.23362	0.01597
	NN-2-layer-droput-input-layer_lr001	0.14103	0.01314
	NN-2-layer-droput-input-layer_lr01	0.23362	0.01597
	NN-2-layer-droput-input-layer_lr1	0.23362	0.01597
	NN-4-layer-droput-each-layer_lr0001	0.16382	0.01397
	NN-4-layer-droput-each-layer_lr01	0.23362	0.01597
	NN-4-layer-droput-each-layer_lr1	0.23362	0.01597
	NN-4-layer_thin_dropout	0.18091	0.01453
	NN-4-layer_thin_dropout_lr01	0.23362	0.01597
	NN-4-layer_thin_dropout_lr1	0.23362	0.01597
	NN-4-layer_wide_no_dropout	0.15670	0.01372
	NN-4-layer_wide_no_dropout_lr01	0.23362	0.01597
	NN-4-layer_wide_no_dropout_lr1	0.23362	0.01597
	NN-4-layer_wide_with_dropout	0.23362	0.01597
	NN-4-layer_wide_with_dropout_lr01	0.23362	0.01597
	NN-4-layer_wide_with_dropout_lr1	0.23362	0.01597
	PassiveAggressiveClassifier	0.11538	0.01206
	RandomForestClassifier	0.07692	0.01006
	SVC	0.08832	0.01071
chess_krvk	BaggingClassifier	0.15520	0.00376
	BaselineClassifier	0.89805	0.00314
	BernoulliNaiveBayes	0.77060	0.00437
	GaussianNaiveBayes	0.76736	0.00439
	GradientBoostingClassifier	0.10692	0.00321
	K_Neighbours	0.30284	0.00478
	NN-12-layer_wide_with_dropout	0.69630	0.00478
	NN-12-layer_wide_with_dropout_lr01	0.87083	0.00349
	NN-12-layer_wide_with_dropout_lr1	0.85301	0.00368
	NN-2-layer-droput-input-layer_lr001	0.77535	0.00434
	NN-2-layer-droput-input-layer_lr01	0.89751	0.00315
	NN-2-layer-droput-input-layer_lr1	0.84016	0.00381
	NN-4-layer-droput-each-layer_lr0001	0.63873	0.00499
	NN-4-layer-droput-each-layer_lr01	0.84016	0.00381
	NN-4-layer-droput-each-layer_lr1	0.89675	0.00316
	NN-4-layer_thin_dropout	0.62782	0.00502
	NN-4-layer_thin_dropout_lr01	0.85301	0.00368
	NN-4-layer_thin_dropout_lr1	0.89048	0.00325
	NN-4-layer_wide_no_dropout	0.54110	0.00518
	NN-4-layer_wide_no_dropout_lr01	0.87083	0.00349
	NN-4-layer_wide_no_dropout_lr1	0.84016	0.00381
	NN-4-layer_wide_with_dropout	0.55319	0.00517
	NN-4-layer_wide_with_dropout_lr01	0.84016	0.00381
	NN-4-layer_wide_with_dropout_lr1	0.85301	0.00368
	PassiveAggressiveClassifier	0.81564	0.00403
	RandomForestClassifier	0.23383	0.00440
	SVC	0.56550	0.00515
chess_krvkp	BaggingClassifier	0.00569	0.00232
	BaselineClassifier	0.51374	0.01539
	BernoulliNaiveBayes	0.13081	0.01038
	GaussianNaiveBayes	0.40569	0.01512
	GradientBoostingClassifier	0.00664	0.00250
	K_Neighbours	0.06161	0.00740

Continued on next page

		loss	std_error
	NN-12-layer_wide_with_dropout	0.16019	0.01129
	NN-12-layer_wide_with_dropout_lr01	0.53555	0.01535
	NN-12-layer_wide_with_dropout_lr1	0.53555	0.01535
	NN-2-layer-droput-input-layer_lr001	0.13649	0.01057
	NN-2-layer-droput-input-layer_lr01	0.46445	0.01535
	NN-2-layer-droput-input-layer_lr1	0.53555	0.01535
	NN-4-layer-droput-each-layer_lr0001	0.05403	0.00696
	NN-4-layer-droput-each-layer_lr01	0.46445	0.01535
	NN-4-layer-droput-each-layer_lr1	0.53555	0.01535
	NN-4-layer_thin_dropout	0.05118	0.00678
	NN-4-layer_thin_dropout_lr01	0.50047	0.01539
	NN-4-layer_thin_dropout_lr1	0.53555	0.01535
	NN-4-layer_wide_no_dropout	0.05498	0.00702
	NN-4-layer_wide_no_dropout_lr01	0.46445	0.01535
	NN-4-layer_wide_no_dropout_lr1	0.53555	0.01535
	NN-4-layer_wide_with_dropout	0.05877	0.00724
	NN-4-layer_wide_with_dropout_lr01	0.53555	0.01535
	NN-4-layer_wide_with_dropout_lr1	0.46445	0.01535
	PassiveAggressiveClassifier	0.02085	0.00440
	RandomForestClassifier	0.00948	0.00298
	SVC	0.00853	0.00283
congressional_voting	BaggingClassifier	0.33333	0.03928
	BaselineClassifier	0.52083	0.04163
	BernoulliNaiveBayes	0.36806	0.04019
	GaussianNaiveBayes	0.47222	0.04160
	GradientBoostingClassifier	0.34028	0.03948
	K_Neighbours	0.34028	0.03948
	NN-12-layer_wide_with_dropout	0.34028	0.03948
	NN-12-layer_wide_with_dropout_lr01	0.34028	0.03948
	NN-12-layer_wide_with_dropout_lr1	0.34028	0.03948
	NN-2-layer-droput-input-layer_lr001	0.34722	0.03967
	NN-2-layer-droput-input-layer_lr01	0.36806	0.04019
	NN-2-layer-droput-input-layer_lr1	0.34028	0.03948
	NN-4-layer-droput-each-layer_lr0001	0.34028	0.03948
	NN-4-layer-droput-each-layer_lr01	0.34028	0.03948
	NN-4-layer-droput-each-layer_lr1	0.34028	0.03948
	NN-4-layer_thin_dropout	0.34028	0.03948
	NN-4-layer_thin_dropout_lr01	0.34028	0.03948
	NN-4-layer_thin_dropout_lr1	0.34722	0.03967
	NN-4-layer_wide_no_dropout	0.40972	0.04098
	NN-4-layer_wide_no_dropout_lr01	0.34028	0.03948
	NN-4-layer_wide_no_dropout_lr1	0.34028	0.03948
	NN-4-layer_wide_with_dropout	0.41667	0.04108
	NN-4-layer_wide_with_dropout_lr01	0.34028	0.03948
	NN-4-layer_wide_with_dropout_lr1	0.34028	0.03948
	PassiveAggressiveClassifier	0.38889	0.04062
	RandomForestClassifier	0.33333	0.03928
	SVC	0.34028	0.03948
conn_bench_sonar_mines_rocks	BaggingClassifier	0.27536	0.05378
	BaselineClassifier	0.57971	0.05942
	BernoulliNaiveBayes	0.23188	0.05081
	GaussianNaiveBayes	0.34783	0.05734
	GradientBoostingClassifier	0.28986	0.05462
	K_Neighbours	0.21739	0.04966
	NN-12-layer_wide_with_dropout	0.49275	0.06019

Continued on next page

		loss	std_error
	NN-12-layer_wide_with_dropout_lr01	0.49275	0.06019
	NN-12-layer_wide_with_dropout_lr1	0.49275	0.06019
	NN-2-layer-droput-input-layer_lr001	0.20290	0.04841
	NN-2-layer-droput-input-layer_lr01	0.49275	0.06019
	NN-2-layer-droput-input-layer_lr1	0.49275	0.06019
	NN-4-layer-droput-each-layer_lr0001	0.34783	0.05734
	NN-4-layer-droput-each-layer_lr01	0.50725	0.06019
	NN-4-layer-droput-each-layer_lr1	0.49275	0.06019
	NN-4-layer_thin_dropout	0.30435	0.05539
	NN-4-layer_thin_dropout_lr01	0.49275	0.06019
	NN-4-layer_thin_dropout_lr1	0.49275	0.06019
	NN-4-layer_wide_no_dropout	0.36232	0.05787
	NN-4-layer_wide_no_dropout_lr01	0.49275	0.06019
	NN-4-layer_wide_no_dropout_lr1	0.50725	0.06019
	NN-4-layer_wide_with_dropout	0.28986	0.05462
	NN-4-layer_wide_with_dropout_lr01	0.49275	0.06019
	NN-4-layer_wide_with_dropout_lr1	0.50725	0.06019
	PassiveAggressiveClassifier	0.23188	0.05081
	RandomForestClassifier	0.17391	0.04563
	SVC	0.24638	0.05187
conn_bench_vowel_deterding	BaggingClassifier	0.04281	0.01119
	BaselineClassifier	0.91131	0.01572
	BernoulliNaiveBayes	0.50765	0.02765
	GaussianNaiveBayes	0.31193	0.02562
	GradientBoostingClassifier	0.08869	0.01572
	K_Neighbours	0.00000	0.00000
	NN-12-layer_wide_with_dropout	0.88991	0.01731
	NN-12-layer_wide_with_dropout_lr01	0.91743	0.01522
	NN-12-layer_wide_with_dropout_lr1	0.90520	0.01620
	NN-2-layer-droput-input-layer_lr001	0.60550	0.02703
	NN-2-layer-droput-input-layer_lr01	0.90520	0.01620
	NN-2-layer-droput-input-layer_lr1	0.90520	0.01620
	NN-4-layer-droput-each-layer_lr0001	0.64220	0.02651
	NN-4-layer-droput-each-layer_lr01	0.91437	0.01547
	NN-4-layer-droput-each-layer_lr1	0.91131	0.01572
	NN-4-layer_thin_dropout	0.67890	0.02582
	NN-4-layer_thin_dropout_lr01	0.91437	0.01547
	NN-4-layer_thin_dropout_lr1	0.90826	0.01596
	NN-4-layer_wide_no_dropout	0.24465	0.02377
	NN-4-layer_wide_no_dropout_lr01	0.91437	0.01547
	NN-4-layer_wide_no_dropout_lr1	0.90520	0.01620
	NN-4-layer_wide_with_dropout	0.35168	0.02641
	NN-4-layer_wide_with_dropout_lr01	0.89602	0.01688
	NN-4-layer_wide_with_dropout_lr1	0.91131	0.01572
	PassiveAggressiveClassifier	0.39144	0.02699
	RandomForestClassifier	0.03670	0.01040
	SVC	0.07645	0.01469
connect_4	BaggingClassifier	0.14282	0.00234
	BaselineClassifier	0.37436	0.00324
	BernoulliNaiveBayes	0.25801	0.00293
	GaussianNaiveBayes	0.32498	0.00314
	GradientBoostingClassifier	0.10505	0.00205
	K_Neighbours	0.17897	0.00257
	NN-12-layer_wide_with_dropout	0.24670	0.00289
	NN-12-layer_wide_with_dropout_lr01	0.24670	0.00289

Continued on next page

		loss	std_error
	NN-12-layer_wide_with_dropout_lr1	0.24670	0.00289
	NN-2-layer-droput-input-layer_lr001	0.25195	0.00291
	NN-2-layer-droput-input-layer_lr01	0.24670	0.00289
	NN-2-layer-droput-input-layer_lr1	0.24670	0.00289
	NN-4-layer-droput-each-layer_lr0001	0.22813	0.00281
	NN-4-layer-droput-each-layer_lr01	0.24670	0.00289
	NN-4-layer-droput-each-layer_lr1	0.24670	0.00289
	NN-4-layer_thin_dropout	0.19866	0.00267
	NN-4-layer_thin_dropout_lr01	0.24670	0.00289
	NN-4-layer_thin_dropout_lr1	0.24670	0.00289
	NN-4-layer_wide_no_dropout	0.24670	0.00289
	NN-4-layer_wide_no_dropout_lr01	0.24670	0.00289
	NN-4-layer_wide_no_dropout_lr1	0.24670	0.00289
	NN-4-layer_wide_with_dropout	0.24670	0.00289
	NN-4-layer_wide_with_dropout_lr01	0.24670	0.00289
	NN-4-layer_wide_with_dropout_lr1	0.24670	0.00289
	PassiveAggressiveClassifier	0.34772	0.00319
	RandomForestClassifier	0.13214	0.00227
	SVC	0.18880	0.00262
contrac	BaggingClassifier	0.45380	0.02256
	BaselineClassifier	0.62012	0.02199
	BernoulliNaiveBayes	0.50924	0.02265
	GaussianNaiveBayes	0.51335	0.02265
	GradientBoostingClassifier	0.46817	0.02261
	K_Neighbours	0.47228	0.02262
	NN-12-layer_wide_with_dropout	0.53799	0.02259
	NN-12-layer_wide_with_dropout_lr01	0.66735	0.02135
	NN-12-layer_wide_with_dropout_lr1	0.56263	0.02248
	NN-2-layer-droput-input-layer_lr001	0.50103	0.02266
	NN-2-layer-droput-input-layer_lr01	0.66735	0.02135
	NN-2-layer-droput-input-layer_lr1	0.66735	0.02135
	NN-4-layer-droput-each-layer_lr0001	0.51129	0.02265
	NN-4-layer-droput-each-layer_lr01	0.56263	0.02248
	NN-4-layer-droput-each-layer_lr1	0.56263	0.02248
	NN-4-layer_thin_dropout	0.48665	0.02265
	NN-4-layer_thin_dropout_lr01	0.66735	0.02135
	NN-4-layer_thin_dropout_lr1	0.56263	0.02248
	NN-4-layer_wide_no_dropout	0.50308	0.02266
	NN-4-layer_wide_no_dropout_lr01	0.56263	0.02248
	NN-4-layer_wide_no_dropout_lr1	0.66735	0.02135
	NN-4-layer_wide_with_dropout	0.49076	0.02265
	NN-4-layer_wide_with_dropout_lr01	0.56263	0.02248
	NN-4-layer_wide_with_dropout_lr1	0.56263	0.02248
	PassiveAggressiveClassifier	0.48255	0.02264
	RandomForestClassifier	0.40657	0.02226
	SVC	0.48871	0.02265
credit_approval	BaggingClassifier	0.15789	0.02415
	BaselineClassifier	0.44737	0.03293
	BernoulliNaiveBayes	0.13596	0.02270
	GaussianNaiveBayes	0.18860	0.02591
	GradientBoostingClassifier	0.16228	0.02442
	K_Neighbours	0.12719	0.02207
	NN-12-layer_wide_with_dropout	0.13596	0.02270
	NN-12-layer_wide_with_dropout_lr01	0.40789	0.03255
	NN-12-layer_wide_with_dropout_lr1	0.59211	0.03255

Continued on next page

		loss	std_error
	NN-2-layer-droput-input-layer_lr001	0.14474	0.02330
	NN-2-layer-droput-input-layer_lr01	0.40789	0.03255
	NN-2-layer-droput-input-layer_lr1	0.37719	0.03210
	NN-4-layer-droput-each-layer_lr0001	0.11404	0.02105
	NN-4-layer-droput-each-layer_lr01	0.40789	0.03255
	NN-4-layer-droput-each-layer_lr1	0.59211	0.03255
	NN-4-layer_thin_dropout	0.12281	0.02174
	NN-4-layer_thin_dropout_lr01	0.59211	0.03255
	NN-4-layer_thin_dropout_lr1	0.40789	0.03255
	NN-4-layer_wide_no_dropout	0.17544	0.02519
	NN-4-layer_wide_no_dropout_lr01	0.59211	0.03255
	NN-4-layer_wide_no_dropout_lr1	0.40789	0.03255
	NN-4-layer_wide_with_dropout	0.23246	0.02797
	NN-4-layer_wide_with_dropout_lr01	0.59211	0.03255
	NN-4-layer_wide_with_dropout_lr1	0.40789	0.03255
	PassiveAggressiveClassifier	0.14474	0.02330
	RandomForestClassifier	0.11842	0.02140
	SVC	0.14035	0.02300
cylinder_bands	BaggingClassifier	0.24260	0.03297
	BaselineClassifier	0.51479	0.03844
	BernoulliNaiveBayes	0.36686	0.03707
	GaussianNaiveBayes	0.35503	0.03681
	GradientBoostingClassifier	0.32544	0.03604
	K_Neighbours	0.24852	0.03324
	NN-12-layer_wide_with_dropout	0.39645	0.03763
	NN-12-layer_wide_with_dropout_lr01	0.39645	0.03763
	NN-12-layer_wide_with_dropout_lr1	0.39645	0.03763
	NN-2-layer-droput-input-layer_lr001	0.33728	0.03637
	NN-2-layer-droput-input-layer_lr01	0.34911	0.03667
	NN-2-layer-droput-input-layer_lr1	0.47929	0.03843
	NN-4-layer-droput-each-layer_lr0001	0.33136	0.03621
	NN-4-layer-droput-each-layer_lr01	0.60355	0.03763
	NN-4-layer-droput-each-layer_lr1	0.39645	0.03763
	NN-4-layer_thin_dropout	0.33136	0.03621
	NN-4-layer_thin_dropout_lr01	0.60355	0.03763
	NN-4-layer_thin_dropout_lr1	0.39645	0.03763
	NN-4-layer_wide_no_dropout	0.34911	0.03667
	NN-4-layer_wide_no_dropout_lr01	0.39645	0.03763
	NN-4-layer_wide_no_dropout_lr1	0.39645	0.03763
	NN-4-layer_wide_with_dropout	0.30769	0.03550
	NN-4-layer_wide_with_dropout_lr01	0.60355	0.03763
	NN-4-layer_wide_with_dropout_lr1	0.43195	0.03810
	PassiveAggressiveClassifier	0.29586	0.03511
	RandomForestClassifier	0.20118	0.03084
	SVC	0.27219	0.03424
dermatology	BaggingClassifier	0.04132	0.01809
	BaselineClassifier	0.80165	0.03625
	BernoulliNaiveBayes	0.04132	0.01809
	GaussianNaiveBayes	0.11570	0.02908
	GradientBoostingClassifier	0.05785	0.02122
	K_Neighbours	0.03306	0.01625
	NN-12-layer_wide_with_dropout	0.70248	0.04156
	NN-12-layer_wide_with_dropout_lr01	0.70248	0.04156
	NN-12-layer_wide_with_dropout_lr1	0.70248	0.04156
	NN-2-layer-droput-input-layer_lr001	0.09917	0.02717

Continued on next page

		loss	std_error
	NN-2-layer-droput-input-layer_lr01	0.62810	0.04394
	NN-2-layer-droput-input-layer_lr1	0.78512	0.03734
	NN-4-layer-droput-each-layer_lr0001	0.19008	0.03567
	NN-4-layer-droput-each-layer_lr01	0.82645	0.03443
	NN-4-layer-droput-each-layer_lr1	0.70248	0.04156
	NN-4-layer_thin_dropout	0.33058	0.04277
	NN-4-layer_thin_dropout_lr01	0.70248	0.04156
	NN-4-layer_thin_dropout_lr1	0.70248	0.04156
	NN-4-layer_wide_no_dropout	0.07438	0.02385
	NN-4-layer_wide_no_dropout_lr01	0.70248	0.04156
	NN-4-layer_wide_no_dropout_lr1	0.52066	0.04542
	NN-4-layer_wide_with_dropout	0.02479	0.01414
	NN-4-layer_wide_with_dropout_lr01	0.70248	0.04156
	NN-4-layer_wide_with_dropout_lr1	0.86777	0.03079
	PassiveAggressiveClassifier	0.02479	0.01414
	RandomForestClassifier	0.03306	0.01625
	SVC	0.02479	0.01414
echocardiogram	BaggingClassifier	0.22727	0.06318
	BaselineClassifier	0.40909	0.07412
	BernoulliNaiveBayes	0.27273	0.06714
	GaussianNaiveBayes	0.20455	0.06081
	GradientBoostingClassifier	0.29545	0.06878
	K_Neighbours	0.20455	0.06081
	NN-12-layer_wide_with_dropout	0.36364	0.07252
	NN-12-layer_wide_with_dropout_lr01	0.36364	0.07252
	NN-12-layer_wide_with_dropout_lr1	0.63636	0.07252
	NN-2-layer-droput-input-layer_lr001	0.25000	0.06528
	NN-2-layer-droput-input-layer_lr01	0.38636	0.07341
	NN-2-layer-droput-input-layer_lr1	0.36364	0.07252
	NN-4-layer-droput-each-layer_lr0001	0.36364	0.07252
	NN-4-layer-droput-each-layer_lr01	0.36364	0.07252
	NN-4-layer-droput-each-layer_lr1	0.36364	0.07252
	NN-4-layer_thin_dropout	0.27273	0.06714
	NN-4-layer_thin_dropout_lr01	0.45455	0.07507
	NN-4-layer_thin_dropout_lr1	0.63636	0.07252
	NN-4-layer_wide_no_dropout	0.27273	0.06714
	NN-4-layer_wide_no_dropout_lr01	0.36364	0.07252
	NN-4-layer_wide_no_dropout_lr1	0.63636	0.07252
	NN-4-layer_wide_with_dropout	0.36364	0.07252
	NN-4-layer_wide_with_dropout_lr01	0.36364	0.07252
	NN-4-layer_wide_with_dropout_lr1	0.63636	0.07252
	PassiveAggressiveClassifier	0.22727	0.06318
	RandomForestClassifier	0.20455	0.06081
	SVC	0.20455	0.06081
ecoli	BaggingClassifier	0.16216	0.03499
	BaselineClassifier	0.63063	0.04581
	BernoulliNaiveBayes	0.17117	0.03575
	GaussianNaiveBayes	0.21622	0.03907
	GradientBoostingClassifier	0.17117	0.03575
	K_Neighbours	0.11712	0.03052
	NN-12-layer_wide_with_dropout	0.56757	0.04702
	NN-12-layer_wide_with_dropout_lr01	0.56757	0.04702
	NN-12-layer_wide_with_dropout_lr1	0.77477	0.03965
	NN-2-layer-droput-input-layer_lr001	0.34234	0.04504
	NN-2-layer-droput-input-layer_lr01	0.35135	0.04531

Continued on next page

		loss	std_error
	NN-2-layer-droput-input-layer_lr1	0.77477	0.03965
	NN-4-layer-droput-each-layer_lr0001	0.34234	0.04504
	NN-4-layer-droput-each-layer_lr01	0.56757	0.04702
	NN-4-layer-droput-each-layer_lr1	0.56757	0.04702
	NN-4-layer_thin_dropout	0.40541	0.04660
	NN-4-layer_thin_dropout_lr01	0.56757	0.04702
	NN-4-layer_thin_dropout_lr1	0.56757	0.04702
	NN-4-layer_wide_no_dropout	0.08108	0.02591
	NN-4-layer_wide_no_dropout_lr01	0.56757	0.04702
	NN-4-layer_wide_no_dropout_lr1	0.56757	0.04702
	NN-4-layer_wide_with_dropout	0.22523	0.03965
	NN-4-layer_wide_with_dropout_lr01	0.77477	0.03965
	NN-4-layer_wide_with_dropout_lr1	0.56757	0.04702
	PassiveAggressiveClassifier	0.09009	0.02718
	RandomForestClassifier	0.16216	0.03499
	SVC	0.11712	0.03052
energy_y1	BaggingClassifier	0.03150	0.01096
	BaselineClassifier	0.62598	0.03036
	BernoulliNaiveBayes	0.20472	0.02532
	GaussianNaiveBayes	0.20472	0.02532
	GradientBoostingClassifier	0.02756	0.01027
	K_Neighbours	0.10630	0.01934
	NN-12-layer_wide_with_dropout	0.27559	0.02804
	NN-12-layer_wide_with_dropout_lr01	0.62598	0.03036
	NN-12-layer_wide_with_dropout_lr1	0.56693	0.03109
	NN-2-layer-droput-input-layer_lr001	0.20472	0.02532
	NN-2-layer-droput-input-layer_lr01	0.20866	0.02550
	NN-2-layer-droput-input-layer_lr1	0.24409	0.02695
	NN-4-layer-droput-each-layer_lr0001	0.19291	0.02476
	NN-4-layer-droput-each-layer_lr01	0.56693	0.03109
	NN-4-layer-droput-each-layer_lr1	0.56693	0.03109
	NN-4-layer_thin_dropout	0.20472	0.02532
	NN-4-layer_thin_dropout_lr01	0.20472	0.02532
	NN-4-layer_thin_dropout_lr1	0.56693	0.03109
	NN-4-layer_wide_no_dropout	0.13386	0.02136
	NN-4-layer_wide_no_dropout_lr01	0.56693	0.03109
	NN-4-layer_wide_no_dropout_lr1	0.62598	0.03036
	NN-4-layer_wide_with_dropout	0.17323	0.02375
	NN-4-layer_wide_with_dropout_lr01	0.56693	0.03109
	NN-4-layer_wide_with_dropout_lr1	0.56693	0.03109
	PassiveAggressiveClassifier	0.19685	0.02495
	RandomForestClassifier	0.02756	0.01027
	SVC	0.06693	0.01568
energy_y2	BaggingClassifier	0.08268	0.01728
	BaselineClassifier	0.59055	0.03085
	BernoulliNaiveBayes	0.12205	0.02054
	GaussianNaiveBayes	0.27953	0.02816
	GradientBoostingClassifier	0.07480	0.01651
	K_Neighbours	0.12205	0.02054
	NN-12-layer_wide_with_dropout	0.27953	0.02816
	NN-12-layer_wide_with_dropout_lr01	0.53543	0.03129
	NN-12-layer_wide_with_dropout_lr1	0.53543	0.03129
	NN-2-layer-droput-input-layer_lr001	0.18504	0.02437
	NN-2-layer-droput-input-layer_lr01	0.29134	0.02851
	NN-2-layer-droput-input-layer_lr1	0.35827	0.03009

Continued on next page

		loss	std_error
	NN-4-layer-droput-each-layer_lr0001	0.17717	0.02396
	NN-4-layer-droput-each-layer_lr01	0.53543	0.03129
	NN-4-layer-droput-each-layer_lr1	0.53543	0.03129
	NN-4-layer_thin_dropout	0.18110	0.02416
	NN-4-layer_thin_dropout_lr01	0.53543	0.03129
	NN-4-layer_thin_dropout_lr1	0.73622	0.02765
	NN-4-layer_wide_no_dropout	0.14567	0.02214
	NN-4-layer_wide_no_dropout_lr01	0.53543	0.03129
	NN-4-layer_wide_no_dropout_lr1	0.53543	0.03129
	NN-4-layer_wide_with_dropout	0.16535	0.02331
	NN-4-layer_wide_with_dropout_lr01	0.53543	0.03129
	NN-4-layer_wide_with_dropout_lr1	0.53543	0.03129
	PassiveAggressiveClassifier	0.14173	0.02188
	RandomForestClassifier	0.08268	0.01728
	SVC	0.08661	0.01765
fertility	BaggingClassifier	0.18182	0.06714
	BaselineClassifier	0.21212	0.07116
	BernoulliNaiveBayes	0.18182	0.06714
	GaussianNaiveBayes	0.18182	0.06714
	GradientBoostingClassifier	0.21212	0.07116
	K_Neighbours	0.18182	0.06714
	NN-12-layer_wide_with_dropout	0.18182	0.06714
	NN-12-layer_wide_with_dropout_lr01	0.18182	0.06714
	NN-12-layer_wide_with_dropout_lr1	0.18182	0.06714
	NN-2-layer-droput-input-layer_lr001	0.18182	0.06714
	NN-2-layer-droput-input-layer_lr01	0.18182	0.06714
	NN-2-layer-droput-input-layer_lr1	0.18182	0.06714
	NN-4-layer-droput-each-layer_lr0001	0.18182	0.06714
	NN-4-layer-droput-each-layer_lr01	0.18182	0.06714
	NN-4-layer-droput-each-layer_lr1	0.18182	0.06714
	NN-4-layer_thin_dropout	0.18182	0.06714
	NN-4-layer_thin_dropout_lr01	0.18182	0.06714
	NN-4-layer_thin_dropout_lr1	0.18182	0.06714
	NN-4-layer_wide_no_dropout	0.18182	0.06714
	NN-4-layer_wide_no_dropout_lr01	0.18182	0.06714
	NN-4-layer_wide_no_dropout_lr1	0.18182	0.06714
	NN-4-layer_wide_with_dropout	0.18182	0.06714
	NN-4-layer_wide_with_dropout_lr01	0.18182	0.06714
	NN-4-layer_wide_with_dropout_lr1	0.18182	0.06714
	PassiveAggressiveClassifier	0.18182	0.06714
	RandomForestClassifier	0.18182	0.06714
	SVC	0.18182	0.06714
flags	BaggingClassifier	0.32308	0.05801
	BaselineClassifier	0.81538	0.04812
	BernoulliNaiveBayes	0.43077	0.06142
	GaussianNaiveBayes	0.72308	0.05550
	GradientBoostingClassifier	0.40000	0.06076
	K_Neighbours	0.44615	0.06166
	NN-12-layer_wide_with_dropout	0.63077	0.05986
	NN-12-layer_wide_with_dropout_lr01	0.63077	0.05986
	NN-12-layer_wide_with_dropout_lr1	0.83077	0.04651
	NN-2-layer-droput-input-layer_lr001	0.64615	0.05931
	NN-2-layer-droput-input-layer_lr01	0.81538	0.04812
	NN-2-layer-droput-input-layer_lr1	0.63077	0.05986
	NN-4-layer-droput-each-layer_lr0001	0.63077	0.05986

Continued on next page

		loss	std_error
	NN-4-layer-droput-each-layer_lr01	0.83077	0.04651
	NN-4-layer-droput-each-layer_lr1	0.63077	0.05986
	NN-4-layer_thin_dropout	0.60000	0.06076
	NN-4-layer_thin_dropout_lr01	0.84615	0.04475
	NN-4-layer_thin_dropout_lr1	0.83077	0.04651
	NN-4-layer_wide_no_dropout	0.56923	0.06142
	NN-4-layer_wide_no_dropout_lr01	0.63077	0.05986
	NN-4-layer_wide_no_dropout_lr1	0.63077	0.05986
	NN-4-layer_wide_with_dropout	0.56923	0.06142
	NN-4-layer_wide_with_dropout_lr01	0.63077	0.05986
	NN-4-layer_wide_with_dropout_lr1	0.63077	0.05986
	PassiveAggressiveClassifier	0.55385	0.06166
	RandomForestClassifier	0.29231	0.05641
	SVC	0.58462	0.06112
glass	BaggingClassifier	0.25352	0.05163
	BaselineClassifier	0.73239	0.05254
	BernoulliNaiveBayes	0.35211	0.05668
	GaussianNaiveBayes	0.73239	0.05254
	GradientBoostingClassifier	0.28169	0.05338
	K_Neighbours	0.25352	0.05163
	NN-12-layer_wide_with_dropout	0.69014	0.05488
	NN-12-layer_wide_with_dropout_lr01	0.69014	0.05488
	NN-12-layer_wide_with_dropout_lr1	0.69014	0.05488
	NN-2-layer-droput-input-layer_lr001	0.61972	0.05761
	NN-2-layer-droput-input-layer_lr01	0.69014	0.05488
	NN-2-layer-droput-input-layer_lr1	0.69014	0.05488
	NN-4-layer-droput-each-layer_lr0001	0.57746	0.05862
	NN-4-layer-droput-each-layer_lr01	0.85915	0.04128
	NN-4-layer-droput-each-layer_lr1	0.69014	0.05488
	NN-4-layer_thin_dropout	0.38028	0.05761
	NN-4-layer_thin_dropout_lr01	0.85915	0.04128
	NN-4-layer_thin_dropout_lr1	0.69014	0.05488
	NN-4-layer_wide_no_dropout	0.36620	0.05717
	NN-4-layer_wide_no_dropout_lr01	0.69014	0.05488
	NN-4-layer_wide_no_dropout_lr1	0.69014	0.05488
	NN-4-layer_wide_with_dropout	0.49296	0.05933
	NN-4-layer_wide_with_dropout_lr01	0.69014	0.05488
	NN-4-layer_wide_with_dropout_lr1	0.61972	0.05761
	PassiveAggressiveClassifier	0.32394	0.05554
	RandomForestClassifier	0.18310	0.04590
	SVC	0.32394	0.05554
haberman_survival	BaggingClassifier	0.32673	0.04667
	BaselineClassifier	0.33663	0.04702
	BernoulliNaiveBayes	0.33663	0.04702
	GaussianNaiveBayes	0.28713	0.04502
	GradientBoostingClassifier	0.38614	0.04844
	K_Neighbours	0.25743	0.04350
	NN-12-layer_wide_with_dropout	0.26733	0.04404
	NN-12-layer_wide_with_dropout_lr01	0.26733	0.04404
	NN-12-layer_wide_with_dropout_lr1	0.26733	0.04404
	NN-2-layer-droput-input-layer_lr001	0.26733	0.04404
	NN-2-layer-droput-input-layer_lr01	0.26733	0.04404
	NN-2-layer-droput-input-layer_lr1	0.26733	0.04404
	NN-4-layer-droput-each-layer_lr0001	0.26733	0.04404
	NN-4-layer-droput-each-layer_lr01	0.26733	0.04404

Continued on next page

		loss	std_error
	NN-4-layer-droput-each-layer_lr1	0.26733	0.04404
	NN-4-layer_thin_dropout	0.26733	0.04404
	NN-4-layer_thin_dropout_lr01	0.26733	0.04404
	NN-4-layer_thin_dropout_lr1	0.26733	0.04404
	NN-4-layer_wide_no_dropout	0.27723	0.04454
	NN-4-layer_wide_no_dropout_lr01	0.26733	0.04404
	NN-4-layer_wide_no_dropout_lr1	0.26733	0.04404
	NN-4-layer_wide_with_dropout	0.29703	0.04547
	NN-4-layer_wide_with_dropout_lr01	0.26733	0.04404
	NN-4-layer_wide_with_dropout_lr1	0.26733	0.04404
	PassiveAggressiveClassifier	0.26733	0.04404
	RandomForestClassifier	0.31683	0.04629
	SVC	0.26733	0.04404
hayes_roth	BaggingClassifier	0.22642	0.05749
	BaselineClassifier	0.60377	0.06718
	BernoulliNaiveBayes	0.47170	0.06857
	GaussianNaiveBayes	0.62264	0.06658
	GradientBoostingClassifier	0.18868	0.05374
	K_Neighbours	0.11321	0.04352
	NN-12-layer_wide_with_dropout	0.54717	0.06837
	NN-12-layer_wide_with_dropout_lr01	0.54717	0.06837
	NN-12-layer_wide_with_dropout_lr1	0.86792	0.04651
	NN-2-layer-droput-input-layer_lr001	0.66038	0.06505
	NN-2-layer-droput-input-layer_lr01	0.58491	0.06768
	NN-2-layer-droput-input-layer_lr1	0.58491	0.06768
	NN-4-layer-droput-each-layer_lr0001	0.58491	0.06768
	NN-4-layer-droput-each-layer_lr01	0.54717	0.06837
	NN-4-layer-droput-each-layer_lr1	0.58491	0.06768
	NN-4-layer_thin_dropout	0.56604	0.06808
	NN-4-layer_thin_dropout_lr01	0.49057	0.06867
	NN-4-layer_thin_dropout_lr1	0.58491	0.06768
	NN-4-layer_wide_no_dropout	0.60377	0.06718
	NN-4-layer_wide_no_dropout_lr01	0.58491	0.06768
	NN-4-layer_wide_no_dropout_lr1	0.86792	0.04651
	NN-4-layer_wide_with_dropout	0.56604	0.06808
	NN-4-layer_wide_with_dropout_lr01	0.58491	0.06768
	NN-4-layer_wide_with_dropout_lr1	0.58491	0.06768
	PassiveAggressiveClassifier	0.58491	0.06768
	RandomForestClassifier	0.15094	0.04917
	SVC	0.18868	0.05374
heart_cleveland	BaggingClassifier	0.33000	0.04702
	BaselineClassifier	0.66000	0.04737
	BernoulliNaiveBayes	0.34000	0.04737
	GaussianNaiveBayes	0.68000	0.04665
	GradientBoostingClassifier	0.46000	0.04984
	K_Neighbours	0.38000	0.04854
	NN-12-layer_wide_with_dropout	0.45000	0.04975
	NN-12-layer_wide_with_dropout_lr01	0.45000	0.04975
	NN-12-layer_wide_with_dropout_lr1	0.45000	0.04975
	NN-2-layer-droput-input-layer_lr001	0.44000	0.04964
	NN-2-layer-droput-input-layer_lr01	0.45000	0.04975
	NN-2-layer-droput-input-layer_lr1	0.45000	0.04975
	NN-4-layer-droput-each-layer_lr0001	0.45000	0.04975
	NN-4-layer-droput-each-layer_lr01	0.45000	0.04975
	NN-4-layer-droput-each-layer_lr1	0.45000	0.04975

Continued on next page

		loss	std_error
	NN-4-layer_thin_dropout	0.55000	0.04975
	NN-4-layer_thin_dropout_lr01	0.45000	0.04975
	NN-4-layer_thin_dropout_lr1	0.45000	0.04975
	NN-4-layer_wide_no_dropout	0.43000	0.04951
	NN-4-layer_wide_no_dropout_lr01	0.45000	0.04975
	NN-4-layer_wide_no_dropout_lr1	0.45000	0.04975
	NN-4-layer_wide_with_dropout	0.35000	0.04770
	NN-4-layer_wide_with_dropout_lr01	0.45000	0.04975
	NN-4-layer_wide_with_dropout_lr1	0.45000	0.04975
	PassiveAggressiveClassifier	0.38000	0.04854
	RandomForestClassifier	0.40000	0.04899
	SVC	0.35000	0.04770
heart_hungarian	BaggingClassifier	0.26531	0.04460
	BaselineClassifier	0.50000	0.05051
	BernoulliNaiveBayes	0.16327	0.03734
	GaussianNaiveBayes	0.18367	0.03911
	GradientBoostingClassifier	0.28571	0.04563
	K_Neighbours	0.16327	0.03734
	NN-12-layer_wide_with_dropout	0.65306	0.04808
	NN-12-layer_wide_with_dropout_lr01	0.34694	0.04808
	NN-12-layer_wide_with_dropout_lr1	0.34694	0.04808
	NN-2-layer-droput-input-layer_lr001	0.17347	0.03825
	NN-2-layer-droput-input-layer_lr01	0.34694	0.04808
	NN-2-layer-droput-input-layer_lr1	0.35714	0.04840
	NN-4-layer-droput-each-layer_lr0001	0.17347	0.03825
	NN-4-layer-droput-each-layer_lr01	0.34694	0.04808
	NN-4-layer-droput-each-layer_lr1	0.34694	0.04808
	NN-4-layer_thin_dropout	0.25510	0.04403
	NN-4-layer_thin_dropout_lr01	0.65306	0.04808
	NN-4-layer_thin_dropout_lr1	0.65306	0.04808
	NN-4-layer_wide_no_dropout	0.19388	0.03993
	NN-4-layer_wide_no_dropout_lr01	0.34694	0.04808
	NN-4-layer_wide_no_dropout_lr1	0.34694	0.04808
	NN-4-layer_wide_with_dropout	0.22449	0.04215
	NN-4-layer_wide_with_dropout_lr01	0.34694	0.04808
	NN-4-layer_wide_with_dropout_lr1	0.34694	0.04808
	PassiveAggressiveClassifier	0.29592	0.04611
	RandomForestClassifier	0.16327	0.03734
	SVC	0.17347	0.03825
heart_switzerland	BaggingClassifier	0.51220	0.07806
	BaselineClassifier	0.68293	0.07267
	BernoulliNaiveBayes	0.60976	0.07618
	GaussianNaiveBayes	0.78049	0.06464
	GradientBoostingClassifier	0.63415	0.07522
	K_Neighbours	0.60976	0.07618
	NN-12-layer_wide_with_dropout	0.75610	0.06707
	NN-12-layer_wide_with_dropout_lr01	0.65854	0.07406
	NN-12-layer_wide_with_dropout_lr1	0.65854	0.07406
	NN-2-layer-droput-input-layer_lr001	0.65854	0.07406
	NN-2-layer-droput-input-layer_lr01	0.65854	0.07406
	NN-2-layer-droput-input-layer_lr1	0.65854	0.07406
	NN-4-layer-droput-each-layer_lr0001	0.65854	0.07406
	NN-4-layer-droput-each-layer_lr01	0.65854	0.07406
	NN-4-layer-droput-each-layer_lr1	0.73171	0.06920
	NN-4-layer_thin_dropout	0.63415	0.07522

Continued on next page

		loss	std_error
	NN-4-layer_thin_dropout_lr01	0.75610	0.06707
	NN-4-layer_thin_dropout_lr1	0.73171	0.06920
	NN-4-layer_wide_no_dropout	0.65854	0.07406
	NN-4-layer_wide_no_dropout_lr01	0.73171	0.06920
	NN-4-layer_wide_no_dropout_lr1	0.65854	0.07406
	NN-4-layer_wide_with_dropout	0.70732	0.07106
	NN-4-layer_wide_with_dropout_lr01	0.65854	0.07406
	NN-4-layer_wide_with_dropout_lr1	0.65854	0.07406
	PassiveAggressiveClassifier	0.63415	0.07522
	RandomForestClassifier	0.58537	0.07694
	SVC	0.60976	0.07618
heart_va	BaggingClassifier	0.77273	0.05158
	BaselineClassifier	0.75758	0.05275
	BernoulliNaiveBayes	0.72727	0.05482
	GaussianNaiveBayes	0.90909	0.03539
	GradientBoostingClassifier	0.71212	0.05573
	K_Neighbours	0.77273	0.05158
	NN-12-layer_wide_with_dropout	0.65152	0.05865
	NN-12-layer_wide_with_dropout_lr01	0.65152	0.05865
	NN-12-layer_wide_with_dropout_lr1	0.89394	0.03790
	NN-2-layer-droput-input-layer_lr001	0.81818	0.04748
	NN-2-layer-droput-input-layer_lr01	0.89394	0.03790
	NN-2-layer-droput-input-layer_lr1	0.65152	0.05865
	NN-4-layer-droput-each-layer_lr0001	0.77273	0.05158
	NN-4-layer-droput-each-layer_lr01	0.89394	0.03790
	NN-4-layer-droput-each-layer_lr1	0.71212	0.05573
	NN-4-layer_thin_dropout	0.75758	0.05275
	NN-4-layer_thin_dropout_lr01	0.78788	0.05032
	NN-4-layer_thin_dropout_lr1	0.89394	0.03790
	NN-4-layer_wide_no_dropout	0.74242	0.05383
	NN-4-layer_wide_no_dropout_lr01	0.89394	0.03790
	NN-4-layer_wide_no_dropout_lr1	0.78788	0.05032
	NN-4-layer_wide_with_dropout	0.66667	0.05803
	NN-4-layer_wide_with_dropout_lr01	0.71212	0.05573
	NN-4-layer_wide_with_dropout_lr1	0.78788	0.05032
	PassiveAggressiveClassifier	0.78788	0.05032
	RandomForestClassifier	0.75758	0.05275
	SVC	0.74242	0.05383
hepatitis	BaggingClassifier	0.15385	0.05003
	BaselineClassifier	0.28846	0.06283
	BernoulliNaiveBayes	0.19231	0.05465
	GaussianNaiveBayes	0.44231	0.06887
	GradientBoostingClassifier	0.28846	0.06283
	K_Neighbours	0.17308	0.05246
	NN-12-layer_wide_with_dropout	0.19231	0.05465
	NN-12-layer_wide_with_dropout_lr01	0.19231	0.05465
	NN-12-layer_wide_with_dropout_lr1	0.19231	0.05465
	NN-2-layer-droput-input-layer_lr001	0.13462	0.04733
	NN-2-layer-droput-input-layer_lr01	0.19231	0.05465
	NN-2-layer-droput-input-layer_lr1	0.19231	0.05465
	NN-4-layer-droput-each-layer_lr0001	0.19231	0.05465
	NN-4-layer-droput-each-layer_lr01	0.19231	0.05465
	NN-4-layer-droput-each-layer_lr1	0.19231	0.05465
	NN-4-layer_thin_dropout	0.25000	0.06005
	NN-4-layer_thin_dropout_lr01	0.19231	0.05465

Continued on next page

		loss	std_error
	NN-4-layer_thin_dropout_lr1	0.19231	0.05465
	NN-4-layer_wide_no_dropout	0.19231	0.05465
	NN-4-layer_wide_no_dropout_lr01	0.19231	0.05465
	NN-4-layer_wide_no_dropout_lr1	0.19231	0.05465
	NN-4-layer_wide_with_dropout	0.15385	0.05003
	NN-4-layer_wide_with_dropout_lr01	0.19231	0.05465
	NN-4-layer_wide_with_dropout_lr1	0.19231	0.05465
	PassiveAggressiveClassifier	0.21154	0.05663
	RandomForestClassifier	0.19231	0.05465
	SVC	0.19231	0.05465
hill_valley	BaggingClassifier	0.42500	0.02472
	BaselineClassifier	0.48500	0.02499
	BernoulliNaiveBayes	0.51000	0.02499
	GaussianNaiveBayes	0.48750	0.02499
	GradientBoostingClassifier	0.45500	0.02490
	K_Neighbours	0.45750	0.02491
	NN-12-layer_wide_with_dropout	0.49250	0.02500
	NN-12-layer_wide_with_dropout_lr01	0.50750	0.02500
	NN-12-layer_wide_with_dropout_lr1	0.49250	0.02500
	NN-2-layer-droput-input-layer_lr001	0.49250	0.02500
	NN-2-layer-droput-input-layer_lr01	0.52250	0.02497
	NN-2-layer-droput-input-layer_lr1	0.49250	0.02500
	NN-4-layer-droput-each-layer_lr0001	0.49250	0.02500
	NN-4-layer-droput-each-layer_lr01	0.49250	0.02500
	NN-4-layer-droput-each-layer_lr1	0.50750	0.02500
	NN-4-layer_thin_dropout	0.52000	0.02498
	NN-4-layer_thin_dropout_lr01	0.49250	0.02500
	NN-4-layer_thin_dropout_lr1	0.49250	0.02500
	NN-4-layer_wide_no_dropout	0.52250	0.02497
	NN-4-layer_wide_no_dropout_lr01	0.49250	0.02500
	NN-4-layer_wide_no_dropout_lr1	0.49250	0.02500
	NN-4-layer_wide_with_dropout	0.52250	0.02497
	NN-4-layer_wide_with_dropout_lr01	0.48750	0.02499
	NN-4-layer_wide_with_dropout_lr1	0.49250	0.02500
	PassiveAggressiveClassifier	0.20250	0.02009
	RandomForestClassifier	0.43750	0.02480
	SVC	0.40000	0.02449
horse_colic	BaggingClassifier	0.12295	0.02973
	BaselineClassifier	0.47541	0.04521
	BernoulliNaiveBayes	0.32787	0.04250
	GaussianNaiveBayes	0.54098	0.04512
	GradientBoostingClassifier	0.18852	0.03541
	K_Neighbours	0.23770	0.03854
	NN-12-layer_wide_with_dropout	0.36066	0.04347
	NN-12-layer_wide_with_dropout_lr01	0.63934	0.04347
	NN-12-layer_wide_with_dropout_lr1	0.36066	0.04347
	NN-2-layer-droput-input-layer_lr001	0.35246	0.04325
	NN-2-layer-droput-input-layer_lr01	0.36066	0.04347
	NN-2-layer-droput-input-layer_lr1	0.36066	0.04347
	NN-4-layer-droput-each-layer_lr0001	0.44262	0.04497
	NN-4-layer-droput-each-layer_lr01	0.36066	0.04347
	NN-4-layer-droput-each-layer_lr1	0.36066	0.04347
	NN-4-layer_thin_dropout	0.41803	0.04466
	NN-4-layer_thin_dropout_lr01	0.36066	0.04347
	NN-4-layer_thin_dropout_lr1	0.36066	0.04347

Continued on next page

		loss	std_error
	NN-4-layer_wide_no_dropout	0.37705	0.04388
	NN-4-layer_wide_no_dropout_lr01	0.36066	0.04347
	NN-4-layer_wide_no_dropout_lr1	0.36066	0.04347
	NN-4-layer_wide_with_dropout	0.41803	0.04466
	NN-4-layer_wide_with_dropout_lr01	0.36066	0.04347
	NN-4-layer_wide_with_dropout_lr1	0.36066	0.04347
	PassiveAggressiveClassifier	0.33607	0.04277
	RandomForestClassifier	0.14754	0.03211
	SVC	0.20492	0.03654
ilpd_indian_liver	BaggingClassifier	0.26943	0.03194
	BaselineClassifier	0.40933	0.03539
	BernoulliNaiveBayes	0.34715	0.03427
	GaussianNaiveBayes	0.40933	0.03539
	GradientBoostingClassifier	0.33679	0.03402
	K_Neighbours	0.29534	0.03284
	NN-12-layer_wide_with_dropout	0.29016	0.03267
	NN-12-layer_wide_with_dropout_lr01	0.29016	0.03267
	NN-12-layer_wide_with_dropout_lr1	0.29016	0.03267
	NN-2-layer-droput-input-layer_lr001	0.29016	0.03267
	NN-2-layer-droput-input-layer_lr01	0.29016	0.03267
	NN-2-layer-droput-input-layer_lr1	0.29016	0.03267
	NN-4-layer-droput-each-layer_lr0001	0.29016	0.03267
	NN-4-layer-droput-each-layer_lr01	0.29016	0.03267
	NN-4-layer-droput-each-layer_lr1	0.29016	0.03267
	NN-4-layer_thin_dropout	0.29016	0.03267
	NN-4-layer_thin_dropout_lr01	0.29016	0.03267
	NN-4-layer_thin_dropout_lr1	0.29016	0.03267
	NN-4-layer_wide_no_dropout	0.31606	0.03347
	NN-4-layer_wide_no_dropout_lr01	0.29016	0.03267
	NN-4-layer_wide_no_dropout_lr1	0.29016	0.03267
	NN-4-layer_wide_with_dropout	0.27979	0.03231
	NN-4-layer_wide_with_dropout_lr01	0.29016	0.03267
	NN-4-layer_wide_with_dropout_lr1	0.29016	0.03267
	PassiveAggressiveClassifier	0.29016	0.03267
	RandomForestClassifier	0.29534	0.03284
	SVC	0.29016	0.03267
image_segmentation	BaggingClassifier	0.03408	0.00657
	BaselineClassifier	0.86370	0.01242
	BernoulliNaiveBayes	0.61992	0.01757
	GaussianNaiveBayes	0.37221	0.01750
	GradientBoostingClassifier	0.04063	0.00715
	K_Neighbours	0.04849	0.00778
	NN-12-layer_wide_with_dropout	0.86501	0.01237
	NN-12-layer_wide_with_dropout_lr01	0.86501	0.01237
	NN-12-layer_wide_with_dropout_lr1	0.86501	0.01237
	NN-2-layer-droput-input-layer_lr001	0.70118	0.01657
	NN-2-layer-droput-input-layer_lr01	0.84928	0.01295
	NN-2-layer-droput-input-layer_lr1	0.86763	0.01227
	NN-4-layer-droput-each-layer_lr0001	0.81520	0.01405
	NN-4-layer-droput-each-layer_lr01	0.84535	0.01309
	NN-4-layer-droput-each-layer_lr1	0.86501	0.01237
	NN-4-layer_thin_dropout	0.78506	0.01487
	NN-4-layer_thin_dropout_lr01	0.86501	0.01237
	NN-4-layer_thin_dropout_lr1	0.86501	0.01237
	NN-4-layer_wide_no_dropout	0.84535	0.01309

Continued on next page

		loss	std_error
	NN-4-layer_wide_no_dropout_lr01	0.86632	0.01232
	NN-4-layer_wide_no_dropout_lr1	0.85714	0.01267
	NN-4-layer_wide_with_dropout	0.78768	0.01480
	NN-4-layer_wide_with_dropout_lr01	0.85714	0.01267
	NN-4-layer_wide_with_dropout_lr1	0.86501	0.01237
	PassiveAggressiveClassifier	0.24509	0.01557
	RandomForestClassifier	0.03408	0.00657
	SVC	0.03539	0.00669
ionosphere	BaggingClassifier	0.13793	0.03202
	BaselineClassifier	0.42241	0.04586
	BernoulliNaiveBayes	0.16379	0.03436
	GaussianNaiveBayes	0.18103	0.03575
	GradientBoostingClassifier	0.14655	0.03284
	K_Neighbours	0.12069	0.03025
	NN-12-layer_wide_with_dropout	0.32759	0.04358
	NN-12-layer_wide_with_dropout_lr01	0.67241	0.04358
	NN-12-layer_wide_with_dropout_lr1	0.32759	0.04358
	NN-2-layer-droput-input-layer_lr001	0.08621	0.02606
	NN-2-layer-droput-input-layer_lr01	0.32759	0.04358
	NN-2-layer-droput-input-layer_lr1	0.22414	0.03872
	NN-4-layer-droput-each-layer_lr0001	0.11207	0.02929
	NN-4-layer-droput-each-layer_lr01	0.32759	0.04358
	NN-4-layer-droput-each-layer_lr1	0.32759	0.04358
	NN-4-layer_thin_dropout	0.08621	0.02606
	NN-4-layer_thin_dropout_lr01	0.32759	0.04358
	NN-4-layer_thin_dropout_lr1	0.32759	0.04358
	NN-4-layer_wide_no_dropout	0.15517	0.03362
	NN-4-layer_wide_no_dropout_lr01	0.27586	0.04150
	NN-4-layer_wide_no_dropout_lr1	0.32759	0.04358
	NN-4-layer_wide_with_dropout	0.12069	0.03025
	NN-4-layer_wide_with_dropout_lr01	0.32759	0.04358
	NN-4-layer_wide_with_dropout_lr1	0.32759	0.04358
	PassiveAggressiveClassifier	0.12069	0.03025
	RandomForestClassifier	0.10345	0.02828
	SVC	0.13793	0.03202
iris	BaggingClassifier	0.04000	0.02771
	BaselineClassifier	0.60000	0.06928
	BernoulliNaiveBayes	0.22000	0.05858
	GaussianNaiveBayes	0.06000	0.03359
	GradientBoostingClassifier	0.04000	0.02771
	K_Neighbours	0.02000	0.01980
	NN-12-layer_wide_with_dropout	0.66000	0.06699
	NN-12-layer_wide_with_dropout_lr01	0.72000	0.06350
	NN-12-layer_wide_with_dropout_lr1	0.66000	0.06699
	NN-2-layer-droput-input-layer_lr001	0.38000	0.06864
	NN-2-layer-droput-input-layer_lr01	0.44000	0.07020
	NN-2-layer-droput-input-layer_lr1	0.72000	0.06350
	NN-4-layer-droput-each-layer_lr0001	0.30000	0.06481
	NN-4-layer-droput-each-layer_lr01	0.66000	0.06699
	NN-4-layer-droput-each-layer_lr1	0.72000	0.06350
	NN-4-layer_thin_dropout	0.36000	0.06788
	NN-4-layer_thin_dropout_lr01	0.26000	0.06203
	NN-4-layer_thin_dropout_lr1	0.72000	0.06350
	NN-4-layer_wide_no_dropout	0.30000	0.06481
	NN-4-layer_wide_no_dropout_lr01	0.38000	0.06864

Continued on next page

		loss	std_error
	NN-4-layer_wide_no_dropout_lr1	0.38000	0.06864
	NN-4-layer_wide_with_dropout	0.14000	0.04907
	NN-4-layer_wide_with_dropout_lr01	0.72000	0.06350
	NN-4-layer_wide_with_dropout_lr1	0.66000	0.06699
	PassiveAggressiveClassifier	0.02000	0.01980
	RandomForestClassifier	0.04000	0.02771
	SVC	0.02000	0.01980
led_display	BaggingClassifier	0.28182	0.02477
	BaselineClassifier	0.91212	0.01559
	BernoulliNaiveBayes	0.28182	0.02477
	GaussianNaiveBayes	0.30909	0.02544
	GradientBoostingClassifier	0.29394	0.02508
	K_Neighbours	0.28788	0.02492
	NN-12-layer_wide_with_dropout	0.86970	0.01853
	NN-12-layer_wide_with_dropout_lr01	0.90000	0.01651
	NN-12-layer_wide_with_dropout_lr1	0.90000	0.01651
	NN-2-layer-droput-input-layer_lr001	0.43030	0.02726
	NN-2-layer-droput-input-layer_lr01	0.85455	0.01941
	NN-2-layer-droput-input-layer_lr1	0.87879	0.01797
	NN-4-layer-droput-each-layer_lr0001	0.33636	0.02601
	NN-4-layer-droput-each-layer_lr01	0.91515	0.01534
	NN-4-layer-droput-each-layer_lr1	0.90000	0.01651
	NN-4-layer_thin_dropout	0.47576	0.02749
	NN-4-layer_thin_dropout_lr01	0.90303	0.01629
	NN-4-layer_thin_dropout_lr1	0.91515	0.01534
	NN-4-layer_wide_no_dropout	0.31515	0.02557
	NN-4-layer_wide_no_dropout_lr01	0.87879	0.01797
	NN-4-layer_wide_no_dropout_lr1	0.90909	0.01583
	NN-4-layer_wide_with_dropout	0.27576	0.02460
	NN-4-layer_wide_with_dropout_lr01	0.92727	0.01430
	NN-4-layer_wide_with_dropout_lr1	0.85455	0.01941
	PassiveAggressiveClassifier	0.27879	0.02468
	RandomForestClassifier	0.29697	0.02515
	SVC	0.27879	0.02468
letter	BaggingClassifier	0.06273	0.00298
	BaselineClassifier	0.96530	0.00225
	BernoulliNaiveBayes	0.58955	0.00606
	GaussianNaiveBayes	0.36091	0.00591
	GradientBoostingClassifier	0.07455	0.00323
	K_Neighbours	0.06106	0.00295
	NN-12-layer_wide_with_dropout	0.91136	0.00350
	NN-12-layer_wide_with_dropout_lr01	0.96288	0.00233
	NN-12-layer_wide_with_dropout_lr1	0.95697	0.00250
	NN-2-layer-droput-input-layer_lr001	0.41515	0.00607
	NN-2-layer-droput-input-layer_lr01	0.93667	0.00300
	NN-2-layer-droput-input-layer_lr1	0.96061	0.00239
	NN-4-layer-droput-each-layer_lr0001	0.40409	0.00604
	NN-4-layer-droput-each-layer_lr01	0.96182	0.00236
	NN-4-layer-droput-each-layer_lr1	0.96227	0.00235
	NN-4-layer_thin_dropout	0.47924	0.00615
	NN-4-layer_thin_dropout_lr01	0.96152	0.00237
	NN-4-layer_thin_dropout_lr1	0.96273	0.00233
	NN-4-layer_wide_no_dropout	0.22727	0.00516
	NN-4-layer_wide_no_dropout_lr01	0.96273	0.00233
	NN-4-layer_wide_no_dropout_lr1	0.96258	0.00234

Continued on next page

		loss	std_error
	NN-4-layer_wide_with_dropout	0.24288	0.00528
	NN-4-layer_wide_with_dropout_lr01	0.96500	0.00226
	NN-4-layer_wide_with_dropout_lr1	0.95864	0.00245
	PassiveAggressiveClassifier	0.35924	0.00591
	RandomForestClassifier	0.04485	0.00255
	SVC	0.04591	0.00258
libras	BaggingClassifier	0.21008	0.03734
	BaselineClassifier	0.95798	0.01839
	BernoulliNaiveBayes	0.49580	0.04583
	GaussianNaiveBayes	0.39496	0.04481
	GradientBoostingClassifier	0.51261	0.04582
	K_Neighbours	0.17647	0.03495
	NN-12-layer_wide_with_dropout	0.84874	0.03285
	NN-12-layer_wide_with_dropout_lr01	0.91597	0.02543
	NN-12-layer_wide_with_dropout_lr1	0.89916	0.02760
	NN-2-layer-droput-input-layer_lr001	0.66387	0.04330
	NN-2-layer-droput-input-layer_lr01	0.90756	0.02655
	NN-2-layer-droput-input-layer_lr1	0.96639	0.01652
	NN-4-layer-droput-each-layer_lr0001	0.69748	0.04211
	NN-4-layer-droput-each-layer_lr01	0.92437	0.02424
	NN-4-layer-droput-each-layer_lr1	0.93277	0.02296
	NN-4-layer_thin_dropout	0.73950	0.04023
	NN-4-layer_thin_dropout_lr01	0.89076	0.02860
	NN-4-layer_thin_dropout_lr1	0.94958	0.02006
	NN-4-layer_wide_no_dropout	0.79832	0.03678
	NN-4-layer_wide_no_dropout_lr01	0.95798	0.01839
	NN-4-layer_wide_no_dropout_lr1	0.95798	0.01839
	NN-4-layer_wide_with_dropout	0.78151	0.03788
	NN-4-layer_wide_with_dropout_lr01	0.93277	0.02296
	NN-4-layer_wide_with_dropout_lr1	0.94958	0.02006
	PassiveAggressiveClassifier	0.37815	0.04445
	RandomForestClassifier	0.25210	0.03980
	SVC	0.23529	0.03888
low_res_spect	BaggingClassifier	0.09091	0.02167
	BaselineClassifier	0.64773	0.03601
	BernoulliNaiveBayes	0.25568	0.03288
	GaussianNaiveBayes	0.20455	0.03041
	GradientBoostingClassifier	0.13636	0.02587
	K_Neighbours	0.10795	0.02339
	NN-12-layer_wide_with_dropout	0.48295	0.03767
	NN-12-layer_wide_with_dropout_lr01	0.48295	0.03767
	NN-12-layer_wide_with_dropout_lr1	0.48295	0.03767
	NN-2-layer-droput-input-layer_lr001	0.24432	0.03239
	NN-2-layer-droput-input-layer_lr01	0.48295	0.03767
	NN-2-layer-droput-input-layer_lr1	0.48295	0.03767
	NN-4-layer-droput-each-layer_lr0001	0.31818	0.03511
	NN-4-layer-droput-each-layer_lr01	0.48295	0.03767
	NN-4-layer-droput-each-layer_lr1	0.48295	0.03767
	NN-4-layer_thin_dropout	0.35795	0.03614
	NN-4-layer_thin_dropout_lr01	0.48295	0.03767
	NN-4-layer_thin_dropout_lr1	0.82386	0.02871
	NN-4-layer_wide_no_dropout	0.22159	0.03131
	NN-4-layer_wide_no_dropout_lr01	0.48295	0.03767
	NN-4-layer_wide_no_dropout_lr1	0.48295	0.03767
	NN-4-layer_wide_with_dropout	0.25568	0.03288

Continued on next page

		loss	std_error
	NN-4-layer_wide_with_dropout_lr01	0.48295	0.03767
	NN-4-layer_wide_with_dropout_lr1	0.48295	0.03767
	PassiveAggressiveClassifier	0.15909	0.02757
	RandomForestClassifier	0.10227	0.02284
	SVC	0.09659	0.02227
lymphography	BaggingClassifier	0.16327	0.05280
	BaselineClassifier	0.61224	0.06961
	BernoulliNaiveBayes	0.26531	0.06307
	GaussianNaiveBayes	0.24490	0.06143
	GradientBoostingClassifier	0.26531	0.06307
	K_Neighbours	0.26531	0.06307
	NN-12-layer_wide_with_dropout	0.65306	0.06800
	NN-12-layer_wide_with_dropout_lr01	0.42857	0.07070
	NN-12-layer_wide_with_dropout_lr1	0.65306	0.06800
	NN-2-layer-droput-input-layer_lr001	0.34694	0.06800
	NN-2-layer-droput-input-layer_lr01	0.46939	0.07129
	NN-2-layer-droput-input-layer_lr1	0.38776	0.06961
	NN-4-layer-droput-each-layer_lr0001	0.30612	0.06584
	NN-4-layer-droput-each-layer_lr01	0.42857	0.07070
	NN-4-layer-droput-each-layer_lr1	0.42857	0.07070
	NN-4-layer_thin_dropout	0.28571	0.06454
	NN-4-layer_thin_dropout_lr01	0.42857	0.07070
	NN-4-layer_thin_dropout_lr1	0.42857	0.07070
	NN-4-layer_wide_no_dropout	0.24490	0.06143
	NN-4-layer_wide_no_dropout_lr01	0.42857	0.07070
	NN-4-layer_wide_no_dropout_lr1	0.42857	0.07070
	NN-4-layer_wide_with_dropout	0.28571	0.06454
	NN-4-layer_wide_with_dropout_lr01	0.65306	0.06800
	NN-4-layer_wide_with_dropout_lr1	0.42857	0.07070
	PassiveAggressiveClassifier	0.20408	0.05758
	RandomForestClassifier	0.20408	0.05758
	SVC	0.20408	0.05758
magic	BaggingClassifier	0.12538	0.00418
	BaselineClassifier	0.45420	0.00628
	BernoulliNaiveBayes	0.24996	0.00547
	GaussianNaiveBayes	0.28071	0.00567
	GradientBoostingClassifier	0.12108	0.00412
	K_Neighbours	0.16393	0.00467
	NN-12-layer_wide_with_dropout	0.65286	0.00601
	NN-12-layer_wide_with_dropout_lr01	0.34714	0.00601
	NN-12-layer_wide_with_dropout_lr1	0.65286	0.00601
	NN-2-layer-droput-input-layer_lr001	0.25761	0.00552
	NN-2-layer-droput-input-layer_lr01	0.34714	0.00601
	NN-2-layer-droput-input-layer_lr1	0.34714	0.00601
	NN-4-layer-droput-each-layer_lr0001	0.14896	0.00449
	NN-4-layer-droput-each-layer_lr01	0.34714	0.00601
	NN-4-layer-droput-each-layer_lr1	0.34714	0.00601
	NN-4-layer_thin_dropout	0.13956	0.00437
	NN-4-layer_thin_dropout_lr01	0.65286	0.00601
	NN-4-layer_thin_dropout_lr1	0.34714	0.00601
	NN-4-layer_wide_no_dropout	0.14513	0.00445
	NN-4-layer_wide_no_dropout_lr01	0.34714	0.00601
	NN-4-layer_wide_no_dropout_lr1	0.34714	0.00601
	NN-4-layer_wide_with_dropout	0.14115	0.00439
	NN-4-layer_wide_with_dropout_lr01	0.34714	0.00601

Continued on next page

		loss	std_error
	NN-4-layer_wide_with_dropout_lr1	0.65286	0.00601
	PassiveAggressiveClassifier	0.22447	0.00527
	RandomForestClassifier	0.12410	0.00416
	SVC	0.17365	0.00478
mammographic	BaggingClassifier	0.23899	0.02392
	BaselineClassifier	0.47799	0.02801
	BernoulliNaiveBayes	0.16352	0.02074
	GaussianNaiveBayes	0.20440	0.02261
	GradientBoostingClassifier	0.23899	0.02392
	K_Neighbours	0.20440	0.02261
	NN-12-layer_wide_with_dropout	0.19497	0.02222
	NN-12-layer_wide_with_dropout_lr01	0.44969	0.02790
	NN-12-layer_wide_with_dropout_lr1	0.44969	0.02790
	NN-2-layer-droput-input-layer_lr001	0.19811	0.02235
	NN-2-layer-droput-input-layer_lr01	0.20126	0.02248
	NN-2-layer-droput-input-layer_lr1	0.21069	0.02287
	NN-4-layer-droput-each-layer_lr0001	0.20755	0.02274
	NN-4-layer-droput-each-layer_lr01	0.44969	0.02790
	NN-4-layer-droput-each-layer_lr1	0.44969	0.02790
	NN-4-layer_thin_dropout	0.20755	0.02274
	NN-4-layer_thin_dropout_lr01	0.24528	0.02413
	NN-4-layer_thin_dropout_lr1	0.55031	0.02790
	NN-4-layer_wide_no_dropout	0.17925	0.02151
	NN-4-layer_wide_no_dropout_lr01	0.55031	0.02790
	NN-4-layer_wide_no_dropout_lr1	0.44969	0.02790
	NN-4-layer_wide_with_dropout	0.17610	0.02136
	NN-4-layer_wide_with_dropout_lr01	0.55031	0.02790
	NN-4-layer_wide_with_dropout_lr1	0.44969	0.02790
	PassiveAggressiveClassifier	0.16981	0.02106
	RandomForestClassifier	0.12893	0.01879
	SVC	0.18239	0.02166
miniboone	BaggingClassifier	0.06554	0.00119
	BaselineClassifier	0.40576	0.00237
	BernoulliNaiveBayes	0.18424	0.00187
	GaussianNaiveBayes	0.71066	0.00219
	GradientBoostingClassifier	0.05838	0.00113
	K_Neighbours	0.10384	0.00147
	NN-12-layer_wide_with_dropout	0.28531	0.00218
	NN-12-layer_wide_with_dropout_lr01	0.28531	0.00218
	NN-12-layer_wide_with_dropout_lr1	0.71469	0.00218
	NN-2-layer-droput-input-layer_lr001	0.17164	0.00182
	NN-2-layer-droput-input-layer_lr01	0.21185	0.00197
	NN-2-layer-droput-input-layer_lr1	0.28531	0.00218
	NN-4-layer-droput-each-layer_lr0001	0.09853	0.00144
	NN-4-layer-droput-each-layer_lr01	0.28531	0.00218
	NN-4-layer-droput-each-layer_lr1	0.28531	0.00218
	NN-4-layer_thin_dropout	0.09606	0.00142
	NN-4-layer_thin_dropout_lr01	0.28531	0.00218
	NN-4-layer_thin_dropout_lr1	0.28531	0.00218
	NN-4-layer_wide_no_dropout	0.09231	0.00140
	NN-4-layer_wide_no_dropout_lr01	0.71469	0.00218
	NN-4-layer_wide_no_dropout_lr1	0.28531	0.00218
	NN-4-layer_wide_with_dropout	0.09431	0.00141
	NN-4-layer_wide_with_dropout_lr01	0.28531	0.00218
	NN-4-layer_wide_with_dropout_lr1	0.28531	0.00218

Continued on next page

		loss	std_error
	PassiveAggressiveClassifier	0.09357	0.00141
	RandomForestClassifier	0.06353	0.00118
	SVC	0.09140	0.00139
molec_biol_promoter	BaggingClassifier	0.08571	0.04732
	BaselineClassifier	0.62857	0.08167
	BernoulliNaiveBayes	0.34286	0.08023
	GaussianNaiveBayes	0.17143	0.06370
	GradientBoostingClassifier	0.25714	0.07388
	K_Neighbours	0.31429	0.07847
	NN-12-layer_wide_with_dropout	0.40000	0.08281
	NN-12-layer_wide_with_dropout_lr01	0.60000	0.08281
	NN-12-layer_wide_with_dropout_lr1	0.60000	0.08281
	NN-2-layer-droput-input-layer_lr001	0.60000	0.08281
	NN-2-layer-droput-input-layer_lr01	0.57143	0.08365
	NN-2-layer-droput-input-layer_lr1	0.60000	0.08281
	NN-4-layer-droput-each-layer_lr0001	0.60000	0.08281
	NN-4-layer-droput-each-layer_lr01	0.60000	0.08281
	NN-4-layer-droput-each-layer_lr1	0.40000	0.08281
	NN-4-layer_thin_dropout	0.45714	0.08420
	NN-4-layer_thin_dropout_lr01	0.42857	0.08365
	NN-4-layer_thin_dropout_lr1	0.40000	0.08281
	NN-4-layer_wide_no_dropout	0.40000	0.08281
	NN-4-layer_wide_no_dropout_lr01	0.60000	0.08281
	NN-4-layer_wide_no_dropout_lr1	0.40000	0.08281
	NN-4-layer_wide_with_dropout	0.40000	0.08281
	NN-4-layer_wide_with_dropout_lr01	0.60000	0.08281
	NN-4-layer_wide_with_dropout_lr1	0.60000	0.08281
	PassiveAggressiveClassifier	0.20000	0.06761
	RandomForestClassifier	0.17143	0.06370
	SVC	0.20000	0.06761
molec_biol_splice	BaggingClassifier	0.06458	0.00757
	BaselineClassifier	0.61633	0.01499
	BernoulliNaiveBayes	0.16144	0.01134
	GaussianNaiveBayes	0.11206	0.00972
	GradientBoostingClassifier	0.04653	0.00649
	K_Neighbours	0.25831	0.01349
	NN-12-layer_wide_with_dropout	0.49098	0.01541
	NN-12-layer_wide_with_dropout_lr01	0.49098	0.01541
	NN-12-layer_wide_with_dropout_lr1	0.49003	0.01541
	NN-2-layer-droput-input-layer_lr001	0.27160	0.01371
	NN-2-layer-droput-input-layer_lr01	0.49098	0.01541
	NN-2-layer-droput-input-layer_lr1	0.49288	0.01541
	NN-4-layer-droput-each-layer_lr0001	0.27445	0.01375
	NN-4-layer-droput-each-layer_lr01	0.49098	0.01541
	NN-4-layer-droput-each-layer_lr1	0.49098	0.01541
	NN-4-layer_thin_dropout	0.34473	0.01465
	NN-4-layer_thin_dropout_lr01	0.49098	0.01541
	NN-4-layer_thin_dropout_lr1	0.49098	0.01541
	NN-4-layer_wide_no_dropout	0.18803	0.01204
	NN-4-layer_wide_no_dropout_lr01	0.49098	0.01541
	NN-4-layer_wide_no_dropout_lr1	0.49098	0.01541
	NN-4-layer_wide_with_dropout	0.22317	0.01283
	NN-4-layer_wide_with_dropout_lr01	0.49098	0.01541
	NN-4-layer_wide_with_dropout_lr1	0.49098	0.01541
	PassiveAggressiveClassifier	0.25641	0.01346

Continued on next page

		loss	std_error
	RandomForestClassifier	0.05128	0.00680
	SVC	0.15480	0.01115
monks_1	BaggingClassifier	0.02174	0.01075
	BaselineClassifier	0.51087	0.03685
	BernoulliNaiveBayes	0.35870	0.03536
	GaussianNaiveBayes	0.33696	0.03485
	GradientBoostingClassifier	0.01087	0.00764
	K_Neighbours	0.19022	0.02893
	NN-12-layer_wide_with_dropout	0.54891	0.03668
	NN-12-layer_wide_with_dropout_lr01	0.54891	0.03668
	NN-12-layer_wide_with_dropout_lr1	0.45109	0.03668
	NN-2-layer-droput-input-layer_lr001	0.58696	0.03630
	NN-2-layer-droput-input-layer_lr01	0.47826	0.03683
	NN-2-layer-droput-input-layer_lr1	0.54891	0.03668
	NN-4-layer-droput-each-layer_lr0001	0.66304	0.03485
	NN-4-layer-droput-each-layer_lr01	0.54891	0.03668
	NN-4-layer-droput-each-layer_lr1	0.54891	0.03668
	NN-4-layer_thin_dropout	0.55978	0.03660
	NN-4-layer_thin_dropout_lr01	0.47826	0.03683
	NN-4-layer_thin_dropout_lr1	0.45109	0.03668
	NN-4-layer_wide_no_dropout	0.57609	0.03643
	NN-4-layer_wide_no_dropout_lr01	0.54891	0.03668
	NN-4-layer_wide_no_dropout_lr1	0.45109	0.03668
	NN-4-layer_wide_with_dropout	0.59239	0.03623
	NN-4-layer_wide_with_dropout_lr01	0.45109	0.03668
	NN-4-layer_wide_with_dropout_lr1	0.54891	0.03668
	PassiveAggressiveClassifier	0.32065	0.03441
	RandomForestClassifier	0.02174	0.01075
	SVC	0.12500	0.02438
monks_3	BaggingClassifier	0.01093	0.00769
	BaselineClassifier	0.51366	0.03695
	BernoulliNaiveBayes	0.22951	0.03109
	GaussianNaiveBayes	0.21858	0.03055
	GradientBoostingClassifier	0.03825	0.01418
	K_Neighbours	0.12568	0.02450
	NN-12-layer_wide_with_dropout	0.45355	0.03680
	NN-12-layer_wide_with_dropout_lr01	0.45355	0.03680
	NN-12-layer_wide_with_dropout_lr1	0.54645	0.03680
	NN-2-layer-droput-input-layer_lr001	0.24044	0.03159
	NN-2-layer-droput-input-layer_lr01	0.44809	0.03676
	NN-2-layer-droput-input-layer_lr1	0.33880	0.03499
	NN-4-layer-droput-each-layer_lr0001	0.21858	0.03055
	NN-4-layer-droput-each-layer_lr01	0.45355	0.03680
	NN-4-layer-droput-each-layer_lr1	0.45355	0.03680
	NN-4-layer_thin_dropout	0.25137	0.03207
	NN-4-layer_thin_dropout_lr01	0.44262	0.03672
	NN-4-layer_thin_dropout_lr1	0.45355	0.03680
	NN-4-layer_wide_no_dropout	0.21858	0.03055
	NN-4-layer_wide_no_dropout_lr01	0.54645	0.03680
	NN-4-layer_wide_no_dropout_lr1	0.45355	0.03680
	NN-4-layer_wide_with_dropout	0.20219	0.02969
	NN-4-layer_wide_with_dropout_lr01	0.54645	0.03680
	NN-4-layer_wide_with_dropout_lr1	0.45355	0.03680
	PassiveAggressiveClassifier	0.21858	0.03055
	RandomForestClassifier	0.01093	0.00769

Continued on next page

		loss	std_error
	SVC	0.07104	0.01899
mushroom	BaggingClassifier	0.00000	0.00000
	BaselineClassifier	0.49273	0.00966
	BernoulliNaiveBayes	0.10481	0.00592
	GaussianNaiveBayes	0.15852	0.00705
	GradientBoostingClassifier	0.00000	0.00000
	K_Neighbours	0.00000	0.00000
	NN-12-layer_wide_with_dropout	0.09735	0.00573
	NN-12-layer_wide_with_dropout_lr01	0.49086	0.00965
	NN-12-layer_wide_with_dropout_lr1	0.49086	0.00965
	NN-2-layer-droput-input-layer_lr001	0.04662	0.00407
	NN-2-layer-droput-input-layer_lr01	0.37150	0.00933
	NN-2-layer-droput-input-layer_lr1	0.49086	0.00965
	NN-4-layer-droput-each-layer_lr0001	0.01529	0.00237
	NN-4-layer-droput-each-layer_lr01	0.50914	0.00965
	NN-4-layer-droput-each-layer_lr1	0.49086	0.00965
	NN-4-layer_thin_dropout	0.00224	0.00091
	NN-4-layer_thin_dropout_lr01	0.49086	0.00965
	NN-4-layer_thin_dropout_lr1	0.49086	0.00965
	NN-4-layer_wide_no_dropout	0.02275	0.00288
	NN-4-layer_wide_no_dropout_lr01	0.16822	0.00722
	NN-4-layer_wide_no_dropout_lr1	0.49086	0.00965
	NN-4-layer_wide_with_dropout	0.05856	0.00453
	NN-4-layer_wide_with_dropout_lr01	0.50914	0.00965
	NN-4-layer_wide_with_dropout_lr1	0.50914	0.00965
	PassiveAggressiveClassifier	0.02909	0.00325
	RandomForestClassifier	0.00000	0.00000
	SVC	0.00112	0.00065
musk_1	BaggingClassifier	0.13291	0.02701
	BaselineClassifier	0.57595	0.03932
	BernoulliNaiveBayes	0.35443	0.03805
	GaussianNaiveBayes	0.29114	0.03614
	GradientBoostingClassifier	0.20886	0.03234
	K_Neighbours	0.14557	0.02806
	NN-12-layer_wide_with_dropout	0.39241	0.03885
	NN-12-layer_wide_with_dropout_lr01	0.60759	0.03885
	NN-12-layer_wide_with_dropout_lr1	0.39241	0.03885
	NN-2-layer-droput-input-layer_lr001	0.24684	0.03430
	NN-2-layer-droput-input-layer_lr01	0.48734	0.03977
	NN-2-layer-droput-input-layer_lr1	0.39241	0.03885
	NN-4-layer-droput-each-layer_lr0001	0.32278	0.03720
	NN-4-layer-droput-each-layer_lr01	0.39241	0.03885
	NN-4-layer-droput-each-layer_lr1	0.60759	0.03885
	NN-4-layer_thin_dropout	0.28481	0.03591
	NN-4-layer_thin_dropout_lr01	0.39241	0.03885
	NN-4-layer_thin_dropout_lr1	0.60759	0.03885
	NN-4-layer_wide_no_dropout	0.25949	0.03487
	NN-4-layer_wide_no_dropout_lr01	0.44304	0.03952
	NN-4-layer_wide_no_dropout_lr1	0.39241	0.03885
	NN-4-layer_wide_with_dropout	0.26582	0.03515
	NN-4-layer_wide_with_dropout_lr01	0.39241	0.03885
	NN-4-layer_wide_with_dropout_lr1	0.39241	0.03885
	PassiveAggressiveClassifier	0.17722	0.03038
	RandomForestClassifier	0.13924	0.02754
	SVC	0.14557	0.02806

Continued on next page

		loss	std_error
musk_2	BaggingClassifier	0.02617	0.00342
	BaselineClassifier	0.25941	0.00939
	BernoulliNaiveBayes	0.26263	0.00943
	GaussianNaiveBayes	0.16345	0.00792
	GradientBoostingClassifier	0.01928	0.00295
	K_Neighbours	0.03489	0.00393
	NN-12-layer_wide_with_dropout	0.14692	0.00759
	NN-12-layer_wide_with_dropout_lr01	0.14692	0.00759
	NN-12-layer_wide_with_dropout_lr1	0.14692	0.00759
	NN-2-layer-droput-input-layer_lr001	0.10790	0.00665
	NN-2-layer-droput-input-layer_lr01	0.14692	0.00759
	NN-2-layer-droput-input-layer_lr1	0.14692	0.00759
	NN-4-layer-droput-each-layer_lr0001	0.14692	0.00759
	NN-4-layer-droput-each-layer_lr01	0.14692	0.00759
	NN-4-layer-droput-each-layer_lr1	0.14692	0.00759
	NN-4-layer_thin_dropout	0.04408	0.00440
	NN-4-layer_thin_dropout_lr01	0.14692	0.00759
	NN-4-layer_thin_dropout_lr1	0.14692	0.00759
	NN-4-layer_wide_no_dropout	0.14692	0.00759
	NN-4-layer_wide_no_dropout_lr01	0.14692	0.00759
	NN-4-layer_wide_no_dropout_lr1	0.14692	0.00759
	NN-4-layer_wide_with_dropout	0.14692	0.00759
	NN-4-layer_wide_with_dropout_lr01	0.14692	0.00759
	NN-4-layer_wide_with_dropout_lr1	0.14692	0.00759
	PassiveAggressiveClassifier	0.06566	0.00531
	RandomForestClassifier	0.02433	0.00330
	SVC	0.02158	0.00311
nursery	BaggingClassifier	0.00257	0.00077
	BaselineClassifier	0.67407	0.00717
	BernoulliNaiveBayes	0.13865	0.00528
	GaussianNaiveBayes	0.27075	0.00679
	GradientBoostingClassifier	0.00210	0.00070
	K_Neighbours	0.04676	0.00323
	NN-12-layer_wide_with_dropout	0.69394	0.00705
	NN-12-layer_wide_with_dropout_lr01	0.65841	0.00725
	NN-12-layer_wide_with_dropout_lr1	0.65841	0.00725
	NN-2-layer-droput-input-layer_lr001	0.11550	0.00489
	NN-2-layer-droput-input-layer_lr01	0.69394	0.00705
	NN-2-layer-droput-input-layer_lr1	0.69394	0.00705
	NN-4-layer-droput-each-layer_lr0001	0.09025	0.00438
	NN-4-layer-droput-each-layer_lr01	0.69394	0.00705
	NN-4-layer-droput-each-layer_lr1	0.65841	0.00725
	NN-4-layer_thin_dropout	0.07435	0.00401
	NN-4-layer_thin_dropout_lr01	0.69394	0.00705
	NN-4-layer_thin_dropout_lr1	0.69394	0.00705
	NN-4-layer_wide_no_dropout	0.06570	0.00379
	NN-4-layer_wide_no_dropout_lr01	0.67314	0.00717
	NN-4-layer_wide_no_dropout_lr1	0.65841	0.00725
	NN-4-layer_wide_with_dropout	0.08464	0.00426
	NN-4-layer_wide_with_dropout_lr01	0.65841	0.00725
	NN-4-layer_wide_with_dropout_lr1	0.65841	0.00725
	PassiveAggressiveClassifier	0.11901	0.00495
	RandomForestClassifier	0.00304	0.00084
	SVC	0.00257	0.00077
oocytes_merluccius_nucleus_4d	BaggingClassifier	0.20414	0.02192

Continued on next page

		loss	std_error
	BaselineClassifier	0.43787	0.02699
	BernoulliNaiveBayes	0.39645	0.02661
	GaussianNaiveBayes	0.40533	0.02670
	GradientBoostingClassifier	0.24556	0.02341
	K_Neighbours	0.25444	0.02369
	NN-12-layer_wide_with_dropout	0.34911	0.02593
	NN-12-layer_wide_with_dropout_lr01	0.34911	0.02593
	NN-12-layer_wide_with_dropout_lr1	0.34911	0.02593
	NN-2-layer-droput-input-layer_lr001	0.32544	0.02549
	NN-2-layer-droput-input-layer_lr01	0.34911	0.02593
	NN-2-layer-droput-input-layer_lr1	0.34911	0.02593
	NN-4-layer-droput-each-layer_lr0001	0.34024	0.02577
	NN-4-layer-droput-each-layer_lr01	0.34911	0.02593
	NN-4-layer-droput-each-layer_lr1	0.34911	0.02593
	NN-4-layer_thin_dropout	0.29882	0.02490
	NN-4-layer_thin_dropout_lr01	0.34911	0.02593
	NN-4-layer_thin_dropout_lr1	0.34911	0.02593
	NN-4-layer_wide_no_dropout	0.26627	0.02404
	NN-4-layer_wide_no_dropout_lr01	0.34911	0.02593
	NN-4-layer_wide_no_dropout_lr1	0.34911	0.02593
	NN-4-layer_wide_with_dropout	0.27811	0.02437
	NN-4-layer_wide_with_dropout_lr01	0.34911	0.02593
	NN-4-layer_wide_with_dropout_lr1	0.34911	0.02593
	PassiveAggressiveClassifier	0.19822	0.02168
	RandomForestClassifier	0.19527	0.02156
	SVC	0.16568	0.02022
oocytes_merluccius_states_2f	BaggingClassifier	0.10059	0.01636
	BaselineClassifier	0.47633	0.02717
	BernoulliNaiveBayes	0.18639	0.02118
	GaussianNaiveBayes	0.14497	0.01915
	GradientBoostingClassifier	0.09467	0.01592
	K_Neighbours	0.08876	0.01547
	NN-12-layer_wide_with_dropout	0.34024	0.02577
	NN-12-layer_wide_with_dropout_lr01	0.34024	0.02577
	NN-12-layer_wide_with_dropout_lr1	0.71598	0.02453
	NN-2-layer-droput-input-layer_lr001	0.13018	0.01830
	NN-2-layer-droput-input-layer_lr01	0.34024	0.02577
	NN-2-layer-droput-input-layer_lr1	0.34024	0.02577
	NN-4-layer-droput-each-layer_lr0001	0.12130	0.01776
	NN-4-layer-droput-each-layer_lr01	0.34024	0.02577
	NN-4-layer-droput-each-layer_lr1	0.34024	0.02577
	NN-4-layer_thin_dropout	0.10947	0.01698
	NN-4-layer_thin_dropout_lr01	0.34024	0.02577
	NN-4-layer_thin_dropout_lr1	0.71598	0.02453
	NN-4-layer_wide_no_dropout	0.14497	0.01915
	NN-4-layer_wide_no_dropout_lr01	0.34024	0.02577
	NN-4-layer_wide_no_dropout_lr1	0.34024	0.02577
	NN-4-layer_wide_with_dropout	0.11538	0.01738
	NN-4-layer_wide_with_dropout_lr01	0.34024	0.02577
	NN-4-layer_wide_with_dropout_lr1	0.34024	0.02577
	PassiveAggressiveClassifier	0.09467	0.01592
	RandomForestClassifier	0.07692	0.01449
	SVC	0.08580	0.01523
oocytes_trisopterus_nucleus_2f	BaggingClassifier	0.19601	0.02288
	BaselineClassifier	0.47176	0.02877

Continued on next page

		loss	std_error
	BernoulliNaiveBayes	0.43189	0.02855
	GaussianNaiveBayes	0.48505	0.02881
	GradientBoostingClassifier	0.20598	0.02331
	K_Neighbours	0.24585	0.02482
	NN-12-layer_wide_with_dropout	0.43189	0.02855
	NN-12-layer_wide_with_dropout_lr01	0.56811	0.02855
	NN-12-layer_wide_with_dropout_lr1	0.43189	0.02855
	NN-2-layer-droput-input-layer_lr001	0.41196	0.02837
	NN-2-layer-droput-input-layer_lr01	0.43189	0.02855
	NN-2-layer-droput-input-layer_lr1	0.43189	0.02855
	NN-4-layer-droput-each-layer_lr0001	0.40199	0.02826
	NN-4-layer-droput-each-layer_lr01	0.43189	0.02855
	NN-4-layer-droput-each-layer_lr1	0.56811	0.02855
	NN-4-layer_thin_dropout	0.34884	0.02747
	NN-4-layer_thin_dropout_lr01	0.43189	0.02855
	NN-4-layer_thin_dropout_lr1	0.43189	0.02855
	NN-4-layer_wide_no_dropout	0.28239	0.02595
	NN-4-layer_wide_no_dropout_lr01	0.43189	0.02855
	NN-4-layer_wide_no_dropout_lr1	0.43189	0.02855
	NN-4-layer_wide_with_dropout	0.30897	0.02663
	NN-4-layer_wide_with_dropout_lr01	0.43189	0.02855
	NN-4-layer_wide_with_dropout_lr1	0.43189	0.02855
	PassiveAggressiveClassifier	0.18605	0.02243
	RandomForestClassifier	0.22591	0.02410
	SVC	0.19601	0.02288
oocytes_trisopterus_states_5b	BaggingClassifier	0.08638	0.01619
	BaselineClassifier	0.47508	0.02878
	BernoulliNaiveBayes	0.23920	0.02459
	GaussianNaiveBayes	0.25249	0.02504
	GradientBoostingClassifier	0.11628	0.01848
	K_Neighbours	0.10963	0.01801
	NN-12-layer_wide_with_dropout	0.55482	0.02865
	NN-12-layer_wide_with_dropout_lr01	0.46179	0.02874
	NN-12-layer_wide_with_dropout_lr1	0.46179	0.02874
	NN-2-layer-droput-input-layer_lr001	0.21927	0.02385
	NN-2-layer-droput-input-layer_lr01	0.55482	0.02865
	NN-2-layer-droput-input-layer_lr1	0.55814	0.02862
	NN-4-layer-droput-each-layer_lr0001	0.20930	0.02345
	NN-4-layer-droput-each-layer_lr01	0.55482	0.02865
	NN-4-layer-droput-each-layer_lr1	0.55482	0.02865
	NN-4-layer_thin_dropout	0.20930	0.02345
	NN-4-layer_thin_dropout_lr01	0.46179	0.02874
	NN-4-layer_thin_dropout_lr1	0.46179	0.02874
	NN-4-layer_wide_no_dropout	0.16279	0.02128
	NN-4-layer_wide_no_dropout_lr01	0.46179	0.02874
	NN-4-layer_wide_no_dropout_lr1	0.46179	0.02874
	NN-4-layer_wide_with_dropout	0.19934	0.02303
	NN-4-layer_wide_with_dropout_lr01	0.55482	0.02865
	NN-4-layer_wide_with_dropout_lr1	0.55482	0.02865
	PassiveAggressiveClassifier	0.10963	0.01801
	RandomForestClassifier	0.08638	0.01619
	SVC	0.07973	0.01561
optical	BaggingClassifier	0.03935	0.00451
	BaselineClassifier	0.89919	0.00699
	BernoulliNaiveBayes	0.10296	0.00706

Continued on next page

		loss	std_error
	GaussianNaiveBayes	0.16388	0.00859
	GradientBoostingClassifier	0.06792	0.00584
	K_Neighbours	0.01725	0.00302
	NN-12-layer_wide_with_dropout	0.89757	0.00704
	NN-12-layer_wide_with_dropout_lr01	0.90566	0.00679
	NN-12-layer_wide_with_dropout_lr1	0.90566	0.00679
	NN-2-layer-droput-input-layer_lr001	0.89811	0.00702
	NN-2-layer-droput-input-layer_lr01	0.89057	0.00725
	NN-2-layer-droput-input-layer_lr1	0.90512	0.00680
	NN-4-layer-droput-each-layer_lr0001	0.89650	0.00707
	NN-4-layer-droput-each-layer_lr01	0.89057	0.00725
	NN-4-layer-droput-each-layer_lr1	0.90189	0.00691
	NN-4-layer_thin_dropout	0.90512	0.00680
	NN-4-layer_thin_dropout_lr01	0.90566	0.00679
	NN-4-layer_thin_dropout_lr1	0.89811	0.00702
	NN-4-layer_wide_no_dropout	0.90081	0.00694
	NN-4-layer_wide_no_dropout_lr01	0.89811	0.00702
	NN-4-layer_wide_no_dropout_lr1	0.89973	0.00697
	NN-4-layer_wide_with_dropout	0.89380	0.00715
	NN-4-layer_wide_with_dropout_lr01	0.90836	0.00670
	NN-4-layer_wide_with_dropout_lr1	0.89865	0.00701
	PassiveAggressiveClassifier	0.04205	0.00466
	RandomForestClassifier	0.01833	0.00311
	SVC	0.01887	0.00316
ozone	BaggingClassifier	0.03106	0.00600
	BaselineClassifier	0.05854	0.00811
	BernoulliNaiveBayes	0.27240	0.01539
	GaussianNaiveBayes	0.28076	0.01553
	GradientBoostingClassifier	0.04182	0.00692
	K_Neighbours	0.02987	0.00588
	NN-12-layer_wide_with_dropout	0.02987	0.00588
	NN-12-layer_wide_with_dropout_lr01	0.02987	0.00588
	NN-12-layer_wide_with_dropout_lr1	0.02987	0.00588
	NN-2-layer-droput-input-layer_lr001	0.02987	0.00588
	NN-2-layer-droput-input-layer_lr01	0.02987	0.00588
	NN-2-layer-droput-input-layer_lr1	0.02987	0.00588
	NN-4-layer-droput-each-layer_lr0001	0.02987	0.00588
	NN-4-layer-droput-each-layer_lr01	0.02987	0.00588
	NN-4-layer-droput-each-layer_lr1	0.02987	0.00588
	NN-4-layer_thin_dropout	0.02987	0.00588
	NN-4-layer_thin_dropout_lr01	0.02987	0.00588
	NN-4-layer_thin_dropout_lr1	0.02987	0.00588
	NN-4-layer_wide_no_dropout	0.02987	0.00588
	NN-4-layer_wide_no_dropout_lr01	0.02987	0.00588
	NN-4-layer_wide_no_dropout_lr1	0.02987	0.00588
	NN-4-layer_wide_with_dropout	0.02987	0.00588
	NN-4-layer_wide_with_dropout_lr01	0.02987	0.00588
	NN-4-layer_wide_with_dropout_lr1	0.02987	0.00588
	PassiveAggressiveClassifier	0.06093	0.00827
	RandomForestClassifier	0.02987	0.00588
	SVC	0.02987	0.00588
page_blocks	BaggingClassifier	0.03431	0.00428
	BaselineClassifier	0.18373	0.00911
	BernoulliNaiveBayes	0.10072	0.00708
	GaussianNaiveBayes	0.06696	0.00588

Continued on next page

		loss	std_error
	GradientBoostingClassifier	0.03431	0.00428
	K_Neighbours	0.03320	0.00421
	NN-12-layer_wide_with_dropout	0.10459	0.00720
	NN-12-layer_wide_with_dropout_lr01	0.10459	0.00720
	NN-12-layer_wide_with_dropout_lr1	0.10459	0.00720
	NN-2-layer-droput-input-layer_lr001	0.08744	0.00665
	NN-2-layer-droput-input-layer_lr01	0.10459	0.00720
	NN-2-layer-droput-input-layer_lr1	0.10459	0.00720
	NN-4-layer-droput-each-layer_lr0001	0.10459	0.00720
	NN-4-layer-droput-each-layer_lr01	0.10459	0.00720
	NN-4-layer-droput-each-layer_lr1	0.10459	0.00720
	NN-4-layer_thin_dropout	0.06641	0.00586
	NN-4-layer_thin_dropout_lr01	0.10459	0.00720
	NN-4-layer_thin_dropout_lr1	0.10459	0.00720
	NN-4-layer_wide_no_dropout	0.10459	0.00720
	NN-4-layer_wide_no_dropout_lr01	0.10459	0.00720
	NN-4-layer_wide_no_dropout_lr1	0.10459	0.00720
	NN-4-layer_wide_with_dropout	0.10459	0.00720
	NN-4-layer_wide_with_dropout_lr01	0.10459	0.00720
	NN-4-layer_wide_with_dropout_lr1	0.10459	0.00720
	PassiveAggressiveClassifier	0.04372	0.00481
	RandomForestClassifier	0.02933	0.00397
	SVC	0.03818	0.00451
parkinsons	BaggingClassifier	0.12308	0.04075
	BaselineClassifier	0.36923	0.05986
	BernoulliNaiveBayes	0.30769	0.05725
	GaussianNaiveBayes	0.32308	0.05801
	GradientBoostingClassifier	0.16923	0.04651
	K_Neighbours	0.09231	0.03590
	NN-12-layer_wide_with_dropout	0.32308	0.05801
	NN-12-layer_wide_with_dropout_lr01	0.32308	0.05801
	NN-12-layer_wide_with_dropout_lr1	0.32308	0.05801
	NN-2-layer-droput-input-layer_lr001	0.30769	0.05725
	NN-2-layer-droput-input-layer_lr01	0.32308	0.05801
	NN-2-layer-droput-input-layer_lr1	0.32308	0.05801
	NN-4-layer-droput-each-layer_lr0001	0.32308	0.05801
	NN-4-layer-droput-each-layer_lr01	0.32308	0.05801
	NN-4-layer-droput-each-layer_lr1	0.32308	0.05801
	NN-4-layer_thin_dropout	0.26154	0.05451
	NN-4-layer_thin_dropout_lr01	0.32308	0.05801
	NN-4-layer_thin_dropout_lr1	0.32308	0.05801
	NN-4-layer_wide_no_dropout	0.32308	0.05801
	NN-4-layer_wide_no_dropout_lr01	0.32308	0.05801
	NN-4-layer_wide_no_dropout_lr1	0.32308	0.05801
	NN-4-layer_wide_with_dropout	0.24615	0.05343
	NN-4-layer_wide_with_dropout_lr01	0.32308	0.05801
	NN-4-layer_wide_with_dropout_lr1	0.32308	0.05801
	PassiveAggressiveClassifier	0.21538	0.05099
	RandomForestClassifier	0.16923	0.04651
	SVC	0.15385	0.04475
pendigits	BaggingClassifier	0.02205	0.00244
	BaselineClassifier	0.89746	0.00504
	BernoulliNaiveBayes	0.19570	0.00659
	GaussianNaiveBayes	0.14774	0.00589
	GradientBoostingClassifier	0.02536	0.00261

Continued on next page

		loss	std_error
	K_Neighbours	0.00606	0.00129
	NN-12-layer_wide_with_dropout	0.89526	0.00508
	NN-12-layer_wide_with_dropout_lr01	0.89746	0.00504
	NN-12-layer_wide_with_dropout_lr1	0.90132	0.00495
	NN-2-layer-droput-input-layer_lr001	0.89443	0.00510
	NN-2-layer-droput-input-layer_lr01	0.90077	0.00496
	NN-2-layer-droput-input-layer_lr1	0.89636	0.00506
	NN-4-layer-droput-each-layer_lr0001	0.89581	0.00507
	NN-4-layer-droput-each-layer_lr01	0.90187	0.00494
	NN-4-layer-droput-each-layer_lr1	0.89746	0.00504
	NN-4-layer_thin_dropout	0.89802	0.00502
	NN-4-layer_thin_dropout_lr01	0.89140	0.00517
	NN-4-layer_thin_dropout_lr1	0.90187	0.00494
	NN-4-layer_wide_no_dropout	0.89774	0.00503
	NN-4-layer_wide_no_dropout_lr01	0.91152	0.00471
	NN-4-layer_wide_no_dropout_lr1	0.90132	0.00495
	NN-4-layer_wide_with_dropout	0.89664	0.00505
	NN-4-layer_wide_with_dropout_lr01	0.90187	0.00494
	NN-4-layer_wide_with_dropout_lr1	0.89140	0.00517
	PassiveAggressiveClassifier	0.07359	0.00433
	RandomForestClassifier	0.01103	0.00173
	SVC	0.01047	0.00169
pima	BaggingClassifier	0.20472	0.02532
	BaselineClassifier	0.49213	0.03137
	BernoulliNaiveBayes	0.24409	0.02695
	GaussianNaiveBayes	0.24016	0.02680
	GradientBoostingClassifier	0.27559	0.02804
	K_Neighbours	0.24409	0.02695
	NN-12-layer_wide_with_dropout	0.36220	0.03016
	NN-12-layer_wide_with_dropout_lr01	0.36220	0.03016
	NN-12-layer_wide_with_dropout_lr1	0.36220	0.03016
	NN-2-layer-droput-input-layer_lr001	0.21654	0.02584
	NN-2-layer-droput-input-layer_lr01	0.36220	0.03016
	NN-2-layer-droput-input-layer_lr1	0.36220	0.03016
	NN-4-layer-droput-each-layer_lr0001	0.20866	0.02550
	NN-4-layer-droput-each-layer_lr01	0.36220	0.03016
	NN-4-layer-droput-each-layer_lr1	0.36220	0.03016
	NN-4-layer_thin_dropout	0.25591	0.02738
	NN-4-layer_thin_dropout_lr01	0.36220	0.03016
	NN-4-layer_thin_dropout_lr1	0.36220	0.03016
	NN-4-layer_wide_no_dropout	0.22441	0.02618
	NN-4-layer_wide_no_dropout_lr01	0.36220	0.03016
	NN-4-layer_wide_no_dropout_lr1	0.36220	0.03016
	NN-4-layer_wide_with_dropout	0.23622	0.02665
	NN-4-layer_wide_with_dropout_lr01	0.63780	0.03016
	NN-4-layer_wide_with_dropout_lr1	0.36220	0.03016
	PassiveAggressiveClassifier	0.19685	0.02495
	RandomForestClassifier	0.20472	0.02532
	SVC	0.21654	0.02584
pittsburg_bridges_MATERIAL	BaggingClassifier	0.14286	0.05915
	BaselineClassifier	0.54286	0.08420
	BernoulliNaiveBayes	0.11429	0.05378
	GaussianNaiveBayes	0.11429	0.05378
	GradientBoostingClassifier	0.14286	0.05915
	K_Neighbours	0.14286	0.05915

Continued on next page

		loss	std_error
	NN-12-layer_wide_with_dropout	0.22857	0.07098
	NN-12-layer_wide_with_dropout_lr01	0.22857	0.07098
	NN-12-layer_wide_with_dropout_lr1	0.22857	0.07098
	NN-2-layer-droput-input-layer_lr001	0.22857	0.07098
	NN-2-layer-droput-input-layer_lr01	0.22857	0.07098
	NN-2-layer-droput-input-layer_lr1	0.22857	0.07098
	NN-4-layer-droput-each-layer_lr0001	0.22857	0.07098
	NN-4-layer-droput-each-layer_lr01	0.22857	0.07098
	NN-4-layer-droput-each-layer_lr1	0.22857	0.07098
	NN-4-layer_thin_dropout	0.37143	0.08167
	NN-4-layer_thin_dropout_lr01	0.22857	0.07098
	NN-4-layer_thin_dropout_lr1	0.22857	0.07098
	NN-4-layer_wide_no_dropout	0.14286	0.05915
	NN-4-layer_wide_no_dropout_lr01	0.22857	0.07098
	NN-4-layer_wide_no_dropout_lr1	0.22857	0.07098
	NN-4-layer_wide_with_dropout	0.20000	0.06761
	NN-4-layer_wide_with_dropout_lr01	0.22857	0.07098
	NN-4-layer_wide_with_dropout_lr1	0.22857	0.07098
	PassiveAggressiveClassifier	0.17143	0.06370
	RandomForestClassifier	0.14286	0.05915
	SVC	0.14286	0.05915
pittsburg_bridges_REL_L	BaggingClassifier	0.47059	0.08560
	BaselineClassifier	0.61765	0.08334
	BernoulliNaiveBayes	0.41176	0.08440
	GaussianNaiveBayes	0.38235	0.08334
	GradientBoostingClassifier	0.55882	0.08515
	K_Neighbours	0.29412	0.07814
	NN-12-layer_wide_with_dropout	0.70588	0.07814
	NN-12-layer_wide_with_dropout_lr01	0.44118	0.08515
	NN-12-layer_wide_with_dropout_lr1	0.44118	0.08515
	NN-2-layer-droput-input-layer_lr001	0.35294	0.08196
	NN-2-layer-droput-input-layer_lr01	0.44118	0.08515
	NN-2-layer-droput-input-layer_lr1	0.35294	0.08196
	NN-4-layer-droput-each-layer_lr0001	0.32353	0.08023
	NN-4-layer-droput-each-layer_lr01	0.44118	0.08515
	NN-4-layer-droput-each-layer_lr1	0.44118	0.08515
	NN-4-layer_thin_dropout	0.41176	0.08440
	NN-4-layer_thin_dropout_lr01	0.70588	0.07814
	NN-4-layer_thin_dropout_lr1	0.44118	0.08515
	NN-4-layer_wide_no_dropout	0.38235	0.08334
	NN-4-layer_wide_no_dropout_lr01	0.44118	0.08515
	NN-4-layer_wide_no_dropout_lr1	0.44118	0.08515
	NN-4-layer_wide_with_dropout	0.29412	0.07814
	NN-4-layer_wide_with_dropout_lr01	0.44118	0.08515
	NN-4-layer_wide_with_dropout_lr1	0.44118	0.08515
	PassiveAggressiveClassifier	0.58824	0.08440
	RandomForestClassifier	0.41176	0.08440
	SVC	0.44118	0.08515
pittsburg_bridges_SPAN	BaggingClassifier	0.41935	0.08863
	BaselineClassifier	0.61290	0.08748
	BernoulliNaiveBayes	0.45161	0.08938
	GaussianNaiveBayes	0.35484	0.08593
	GradientBoostingClassifier	0.32258	0.08396
	K_Neighbours	0.22581	0.07510
	NN-12-layer_wide_with_dropout	0.38710	0.08748

Continued on next page

		loss	std_error
	NN-12-layer_wide_with_dropout_lr01	0.38710	0.08748
	NN-12-layer_wide_with_dropout_lr1	0.38710	0.08748
	NN-2-layer-droput-input-layer_lr001	0.51613	0.08976
	NN-2-layer-droput-input-layer_lr01	0.45161	0.08938
	NN-2-layer-droput-input-layer_lr1	0.38710	0.08748
	NN-4-layer-droput-each-layer_lr0001	0.38710	0.08748
	NN-4-layer-droput-each-layer_lr01	0.38710	0.08748
	NN-4-layer-droput-each-layer_lr1	0.38710	0.08748
	NN-4-layer_thin_dropout	0.64516	0.08593
	NN-4-layer_thin_dropout_lr01	0.48387	0.08976
	NN-4-layer_thin_dropout_lr1	0.38710	0.08748
	NN-4-layer_wide_no_dropout	0.45161	0.08938
	NN-4-layer_wide_no_dropout_lr01	0.38710	0.08748
	NN-4-layer_wide_no_dropout_lr1	0.38710	0.08748
	NN-4-layer_wide_with_dropout	0.16129	0.06606
	NN-4-layer_wide_with_dropout_lr01	0.74194	0.07859
	NN-4-layer_wide_with_dropout_lr1	0.83871	0.06606
	PassiveAggressiveClassifier	0.41935	0.08863
	RandomForestClassifier	0.19355	0.07096
	SVC	0.32258	0.08396
pittsburg_bridges_TYPE	BaggingClassifier	0.42857	0.08365
	BaselineClassifier	0.80000	0.06761
	BernoulliNaiveBayes	0.42857	0.08365
	GaussianNaiveBayes	0.54286	0.08420
	GradientBoostingClassifier	0.42857	0.08365
	K_Neighbours	0.45714	0.08420
	NN-12-layer_wide_with_dropout	0.51429	0.08448
	NN-12-layer_wide_with_dropout_lr01	0.51429	0.08448
	NN-12-layer_wide_with_dropout_lr1	0.51429	0.08448
	NN-2-layer-droput-input-layer_lr001	0.54286	0.08420
	NN-2-layer-droput-input-layer_lr01	0.71429	0.07636
	NN-2-layer-droput-input-layer_lr1	0.42857	0.08365
	NN-4-layer-droput-each-layer_lr0001	0.48571	0.08448
	NN-4-layer-droput-each-layer_lr01	0.51429	0.08448
	NN-4-layer-droput-each-layer_lr1	0.51429	0.08448
	NN-4-layer_thin_dropout	0.51429	0.08448
	NN-4-layer_thin_dropout_lr01	0.45714	0.08420
	NN-4-layer_thin_dropout_lr1	0.51429	0.08448
	NN-4-layer_wide_no_dropout	0.51429	0.08448
	NN-4-layer_wide_no_dropout_lr01	0.51429	0.08448
	NN-4-layer_wide_no_dropout_lr1	0.51429	0.08448
	NN-4-layer_wide_with_dropout	0.45714	0.08420
	NN-4-layer_wide_with_dropout_lr01	0.51429	0.08448
	NN-4-layer_wide_with_dropout_lr1	0.51429	0.08448
	PassiveAggressiveClassifier	0.51429	0.08448
	RandomForestClassifier	0.40000	0.08281
	SVC	0.40000	0.08281
pittsburg_bridges_T_OR_D	BaggingClassifier	0.29412	0.07814
	BaselineClassifier	0.26471	0.07566
	BernoulliNaiveBayes	0.23529	0.07275
	GaussianNaiveBayes	0.20588	0.06934
	GradientBoostingClassifier	0.32353	0.08023
	K_Neighbours	0.26471	0.07566
	NN-12-layer_wide_with_dropout	0.20588	0.06934
	NN-12-layer_wide_with_dropout_lr01	0.20588	0.06934

Continued on next page

		loss	std_error
	NN-12-layer_wide_with_dropout_lr1	0.20588	0.06934
	NN-2-layer-droput-input-layer_lr001	0.20588	0.06934
	NN-2-layer-droput-input-layer_lr01	0.20588	0.06934
	NN-2-layer-droput-input-layer_lr1	0.20588	0.06934
	NN-4-layer-droput-each-layer_lr0001	0.20588	0.06934
	NN-4-layer-droput-each-layer_lr01	0.20588	0.06934
	NN-4-layer-droput-each-layer_lr1	0.20588	0.06934
	NN-4-layer_thin_dropout	0.20588	0.06934
	NN-4-layer_thin_dropout_lr01	0.20588	0.06934
	NN-4-layer_thin_dropout_lr1	0.20588	0.06934
	NN-4-layer_wide_no_dropout	0.20588	0.06934
	NN-4-layer_wide_no_dropout_lr01	0.20588	0.06934
	NN-4-layer_wide_no_dropout_lr1	0.20588	0.06934
	NN-4-layer_wide_with_dropout	0.20588	0.06934
	NN-4-layer_wide_with_dropout_lr01	0.20588	0.06934
	NN-4-layer_wide_with_dropout_lr1	0.20588	0.06934
	PassiveAggressiveClassifier	0.29412	0.07814
	RandomForestClassifier	0.26471	0.07566
	SVC	0.23529	0.07275
planning	BaggingClassifier	0.39344	0.06255
	BaselineClassifier	0.44262	0.06360
	BernoulliNaiveBayes	0.31148	0.05929
	GaussianNaiveBayes	0.44262	0.06360
	GradientBoostingClassifier	0.45902	0.06380
	K_Neighbours	0.29508	0.05840
	NN-12-layer_wide_with_dropout	0.27869	0.05741
	NN-12-layer_wide_with_dropout_lr01	0.27869	0.05741
	NN-12-layer_wide_with_dropout_lr1	0.27869	0.05741
	NN-2-layer-droput-input-layer_lr001	0.26230	0.05632
	NN-2-layer-droput-input-layer_lr01	0.27869	0.05741
	NN-2-layer-droput-input-layer_lr1	0.27869	0.05741
	NN-4-layer-droput-each-layer_lr0001	0.27869	0.05741
	NN-4-layer-droput-each-layer_lr01	0.27869	0.05741
	NN-4-layer-droput-each-layer_lr1	0.27869	0.05741
	NN-4-layer_thin_dropout	0.27869	0.05741
	NN-4-layer_thin_dropout_lr01	0.27869	0.05741
	NN-4-layer_thin_dropout_lr1	0.27869	0.05741
	NN-4-layer_wide_no_dropout	0.27869	0.05741
	NN-4-layer_wide_no_dropout_lr01	0.27869	0.05741
	NN-4-layer_wide_no_dropout_lr1	0.27869	0.05741
	NN-4-layer_wide_with_dropout	0.27869	0.05741
	NN-4-layer_wide_with_dropout_lr01	0.27869	0.05741
	NN-4-layer_wide_with_dropout_lr1	0.27869	0.05741
	PassiveAggressiveClassifier	0.36066	0.06148
	RandomForestClassifier	0.34426	0.06083
	SVC	0.27869	0.05741
plant_margin	BaggingClassifier	0.22348	0.01813
	BaselineClassifier	0.98674	0.00498
	BernoulliNaiveBayes	0.33523	0.02054
	GaussianNaiveBayes	0.29735	0.01989
	GradientBoostingClassifier	0.59848	0.02133
	K_Neighbours	0.21591	0.01791
	NN-12-layer_wide_with_dropout	0.99242	0.00377
	NN-12-layer_wide_with_dropout_lr01	0.99053	0.00421
	NN-12-layer_wide_with_dropout_lr1	0.99621	0.00267

Continued on next page

		loss	std_error
	NN-2-layer-droput-input-layer_lr001	0.87879	0.01420
	NN-2-layer-droput-input-layer_lr01	0.99621	0.00267
	NN-2-layer-droput-input-layer_lr1	0.99432	0.00327
	NN-4-layer-droput-each-layer_lr0001	0.98485	0.00532
	NN-4-layer-droput-each-layer_lr01	0.99053	0.00421
	NN-4-layer-droput-each-layer_lr1	0.98864	0.00461
	NN-4-layer_thin_dropout	0.95644	0.00888
	NN-4-layer_thin_dropout_lr01	0.98674	0.00498
	NN-4-layer_thin_dropout_lr1	0.99053	0.00421
	NN-4-layer_wide_no_dropout	0.84659	0.01568
	NN-4-layer_wide_no_dropout_lr01	0.98674	0.00498
	NN-4-layer_wide_no_dropout_lr1	0.99053	0.00421
	NN-4-layer_wide_with_dropout	0.80871	0.01712
	NN-4-layer_wide_with_dropout_lr01	0.99053	0.00421
	NN-4-layer_wide_with_dropout_lr1	0.99053	0.00421
	PassiveAggressiveClassifier	0.43939	0.02160
	RandomForestClassifier	0.20455	0.01755
	SVC	0.19318	0.01718
plant_shape	BaggingClassifier	0.42614	0.02152
	BaselineClassifier	0.98674	0.00498
	BernoulliNaiveBayes	0.82386	0.01658
	GaussianNaiveBayes	0.46780	0.02171
	GradientBoostingClassifier	0.68939	0.02014
	K_Neighbours	0.41288	0.02143
	NN-12-layer_wide_with_dropout	0.99053	0.00421
	NN-12-layer_wide_with_dropout_lr01	0.99053	0.00421
	NN-12-layer_wide_with_dropout_lr1	0.98864	0.00461
	NN-2-layer-droput-input-layer_lr001	0.96212	0.00831
	NN-2-layer-droput-input-layer_lr01	1.00000	0.00000
	NN-2-layer-droput-input-layer_lr1	0.99242	0.00377
	NN-4-layer-droput-each-layer_lr0001	0.97348	0.00699
	NN-4-layer-droput-each-layer_lr01	0.99242	0.00377
	NN-4-layer-droput-each-layer_lr1	0.98864	0.00461
	NN-4-layer_thin_dropout	0.97159	0.00723
	NN-4-layer_thin_dropout_lr01	0.99242	0.00377
	NN-4-layer_thin_dropout_lr1	0.99432	0.00327
	NN-4-layer_wide_no_dropout	0.95265	0.00924
	NN-4-layer_wide_no_dropout_lr01	0.99053	0.00421
	NN-4-layer_wide_no_dropout_lr1	0.98674	0.00498
	NN-4-layer_wide_with_dropout	0.94886	0.00959
	NN-4-layer_wide_with_dropout_lr01	0.99242	0.00377
	NN-4-layer_wide_with_dropout_lr1	0.99053	0.00421
	PassiveAggressiveClassifier	0.61174	0.02121
	RandomForestClassifier	0.41098	0.02141
	SVC	0.32008	0.02030
plant_texture	BaggingClassifier	0.24053	0.01860
	BaselineClassifier	0.98106	0.00593
	BernoulliNaiveBayes	0.31250	0.02017
	GaussianNaiveBayes	0.38258	0.02115
	GradientBoostingClassifier	0.50947	0.02176
	K_Neighbours	0.19318	0.01718
	NN-12-layer_wide_with_dropout	0.99242	0.00377
	NN-12-layer_wide_with_dropout_lr01	0.98864	0.00461
	NN-12-layer_wide_with_dropout_lr1	0.99242	0.00377
	NN-2-layer-droput-input-layer_lr001	0.85795	0.01519

Continued on next page

		loss	std_error
	NN-2-layer-droput-input-layer_lr01	0.96970	0.00746
	NN-2-layer-droput-input-layer_lr1	0.99053	0.00421
	NN-4-layer-droput-each-layer_lr0001	0.97727	0.00649
	NN-4-layer-droput-each-layer_lr01	0.98295	0.00563
	NN-4-layer-droput-each-layer_lr1	0.99053	0.00421
	NN-4-layer_thin_dropout	0.95833	0.00870
	NN-4-layer_thin_dropout_lr01	0.98485	0.00532
	NN-4-layer_thin_dropout_lr1	0.99432	0.00327
	NN-4-layer_wide_no_dropout	0.82197	0.01665
	NN-4-layer_wide_no_dropout_lr01	0.99242	0.00377
	NN-4-layer_wide_no_dropout_lr1	0.99432	0.00327
	NN-4-layer_wide_with_dropout	0.87500	0.01439
	NN-4-layer_wide_with_dropout_lr01	0.99242	0.00377
	NN-4-layer_wide_with_dropout_lr1	0.98674	0.00498
	PassiveAggressiveClassifier	0.34848	0.02074
	RandomForestClassifier	0.17992	0.01672
	SVC	0.17992	0.01672
post_operative	BaggingClassifier	0.36667	0.08798
	BaselineClassifier	0.36667	0.08798
	BernoulliNaiveBayes	0.30000	0.08367
	GaussianNaiveBayes	0.26667	0.08074
	GradientBoostingClassifier	0.43333	0.09047
	K_Neighbours	0.26667	0.08074
	NN-12-layer_wide_with_dropout	0.26667	0.08074
	NN-12-layer_wide_with_dropout_lr01	0.26667	0.08074
	NN-12-layer_wide_with_dropout_lr1	0.26667	0.08074
	NN-2-layer-droput-input-layer_lr001	0.30000	0.08367
	NN-2-layer-droput-input-layer_lr01	0.26667	0.08074
	NN-2-layer-droput-input-layer_lr1	0.26667	0.08074
	NN-4-layer-droput-each-layer_lr0001	0.26667	0.08074
	NN-4-layer-droput-each-layer_lr01	0.26667	0.08074
	NN-4-layer-droput-each-layer_lr1	0.26667	0.08074
	NN-4-layer_thin_dropout	0.26667	0.08074
	NN-4-layer_thin_dropout_lr01	0.26667	0.08074
	NN-4-layer_thin_dropout_lr1	0.26667	0.08074
	NN-4-layer_wide_no_dropout	0.26667	0.08074
	NN-4-layer_wide_no_dropout_lr01	0.26667	0.08074
	NN-4-layer_wide_no_dropout_lr1	0.26667	0.08074
	NN-4-layer_wide_with_dropout	0.26667	0.08074
	NN-4-layer_wide_with_dropout_lr01	0.26667	0.08074
	NN-4-layer_wide_with_dropout_lr1	0.26667	0.08074
	PassiveAggressiveClassifier	0.30000	0.08367
	RandomForestClassifier	0.33333	0.08607
	SVC	0.26667	0.08074
primary_tumor	BaggingClassifier	0.60550	0.04681
	BaselineClassifier	0.86239	0.03300
	BernoulliNaiveBayes	0.55046	0.04765
	GaussianNaiveBayes	0.81651	0.03707
	GradientBoostingClassifier	0.57798	0.04731
	K_Neighbours	0.53211	0.04779
	NN-12-layer_wide_with_dropout	0.77982	0.03969
	NN-12-layer_wide_with_dropout_lr01	0.77982	0.03969
	NN-12-layer_wide_with_dropout_lr1	0.77982	0.03969
	NN-2-layer-droput-input-layer_lr001	0.70642	0.04362
	NN-2-layer-droput-input-layer_lr01	0.77982	0.03969

Continued on next page

		loss	std_error
	NN-2-layer-droput-input-layer_lr1	0.77982	0.03969
	NN-4-layer-droput-each-layer_lr0001	0.77982	0.03969
	NN-4-layer-droput-each-layer_lr01	0.77982	0.03969
	NN-4-layer-droput-each-layer_lr1	0.77982	0.03969
	NN-4-layer_thin_dropout	0.76147	0.04082
	NN-4-layer_thin_dropout_lr01	0.77982	0.03969
	NN-4-layer_thin_dropout_lr1	0.77982	0.03969
	NN-4-layer_wide_no_dropout	0.58716	0.04716
	NN-4-layer_wide_no_dropout_lr01	0.77982	0.03969
	NN-4-layer_wide_no_dropout_lr1	0.77982	0.03969
	NN-4-layer_wide_with_dropout	0.57798	0.04731
	NN-4-layer_wide_with_dropout_lr01	0.77982	0.03969
	NN-4-layer_wide_with_dropout_lr1	0.90826	0.02765
	PassiveAggressiveClassifier	0.61468	0.04661
	RandomForestClassifier	0.55046	0.04765
	SVC	0.56881	0.04744
ringnorm	BaggingClassifier	0.04300	0.00410
	BaselineClassifier	0.51679	0.01011
	BernoulliNaiveBayes	0.25635	0.00884
	GaussianNaiveBayes	0.01106	0.00212
	GradientBoostingClassifier	0.03194	0.00356
	K_Neighbours	0.20229	0.00813
	NN-12-layer_wide_with_dropout	0.03849	0.00389
	NN-12-layer_wide_with_dropout_lr01	0.49304	0.01012
	NN-12-layer_wide_with_dropout_lr1	0.49304	0.01012
	NN-2-layer-droput-input-layer_lr001	0.17813	0.00774
	NN-2-layer-droput-input-layer_lr01	0.28215	0.00911
	NN-2-layer-droput-input-layer_lr1	0.62490	0.00980
	NN-4-layer-droput-each-layer_lr0001	0.06839	0.00511
	NN-4-layer-droput-each-layer_lr01	0.49304	0.01012
	NN-4-layer-droput-each-layer_lr1	0.49304	0.01012
	NN-4-layer_thin_dropout	0.08149	0.00554
	NN-4-layer_thin_dropout_lr01	0.49304	0.01012
	NN-4-layer_thin_dropout_lr1	0.49304	0.01012
	NN-4-layer_wide_no_dropout	0.03030	0.00347
	NN-4-layer_wide_no_dropout_lr01	0.50696	0.01012
	NN-4-layer_wide_no_dropout_lr1	0.48853	0.01012
	NN-4-layer_wide_with_dropout	0.05242	0.00451
	NN-4-layer_wide_with_dropout_lr01	0.49304	0.01012
	NN-4-layer_wide_with_dropout_lr1	0.50696	0.01012
	PassiveAggressiveClassifier	0.24283	0.00868
	RandomForestClassifier	0.03767	0.00385
	SVC	0.01925	0.00278
seeds	BaggingClassifier	0.04286	0.02421
	BaselineClassifier	0.68571	0.05549
	BernoulliNaiveBayes	0.20000	0.04781
	GaussianNaiveBayes	0.07143	0.03078
	GradientBoostingClassifier	0.02857	0.01991
	K_Neighbours	0.08571	0.03346
	NN-12-layer_wide_with_dropout	0.61429	0.05818
	NN-12-layer_wide_with_dropout_lr01	0.72857	0.05315
	NN-12-layer_wide_with_dropout_lr1	0.65714	0.05673
	NN-2-layer-droput-input-layer_lr001	0.37143	0.05775
	NN-2-layer-droput-input-layer_lr01	0.38571	0.05818
	NN-2-layer-droput-input-layer_lr1	0.64286	0.05727

Continued on next page

		loss	std_error
	NN-4-layer-droput-each-layer_lr0001	0.22857	0.05019
	NN-4-layer-droput-each-layer_lr01	0.72857	0.05315
	NN-4-layer-droput-each-layer_lr1	0.72857	0.05315
	NN-4-layer_thin_dropout	0.38571	0.05818
	NN-4-layer_thin_dropout_lr01	0.38571	0.05818
	NN-4-layer_thin_dropout_lr1	0.61429	0.05818
	NN-4-layer_wide_no_dropout	0.20000	0.04781
	NN-4-layer_wide_no_dropout_lr01	0.50000	0.05976
	NN-4-layer_wide_no_dropout_lr1	0.72857	0.05315
	NN-4-layer_wide_with_dropout	0.18571	0.04648
	NN-4-layer_wide_with_dropout_lr01	0.65714	0.05673
	NN-4-layer_wide_with_dropout_lr1	0.72857	0.05315
	PassiveAggressiveClassifier	0.04286	0.02421
	RandomForestClassifier	0.04286	0.02421
	SVC	0.05714	0.02774
soybean	BaggingClassifier	0.07965	0.01801
	BaselineClassifier	0.90265	0.01972
	BernoulliNaiveBayes	0.35398	0.03181
	GaussianNaiveBayes	0.20796	0.02700
	GradientBoostingClassifier	0.09735	0.01972
	K_Neighbours	0.11947	0.02157
	NN-12-layer_wide_with_dropout	0.86283	0.02288
	NN-12-layer_wide_with_dropout_lr01	0.86726	0.02257
	NN-12-layer_wide_with_dropout_lr1	0.86283	0.02288
	NN-2-layer-droput-input-layer_lr001	0.69912	0.03051
	NN-2-layer-droput-input-layer_lr01	0.89823	0.02011
	NN-2-layer-droput-input-layer_lr1	0.89823	0.02011
	NN-4-layer-droput-each-layer_lr0001	0.78761	0.02721
	NN-4-layer-droput-each-layer_lr01	0.89823	0.02011
	NN-4-layer-droput-each-layer_lr1	0.80531	0.02634
	NN-4-layer_thin_dropout	0.84513	0.02407
	NN-4-layer_thin_dropout_lr01	0.86283	0.02288
	NN-4-layer_thin_dropout_lr1	0.89823	0.02011
	NN-4-layer_wide_no_dropout	0.74779	0.02889
	NN-4-layer_wide_no_dropout_lr01	0.89823	0.02011
	NN-4-layer_wide_no_dropout_lr1	0.89823	0.02011
	NN-4-layer_wide_with_dropout	0.71681	0.02997
	NN-4-layer_wide_with_dropout_lr01	0.86726	0.02257
	NN-4-layer_wide_with_dropout_lr1	0.86283	0.02288
	PassiveAggressiveClassifier	0.09292	0.01931
	RandomForestClassifier	0.07965	0.01801
	SVC	0.06637	0.01656
spambase	BaggingClassifier	0.05464	0.00583
	BaselineClassifier	0.48387	0.01282
	BernoulliNaiveBayes	0.10336	0.00781
	GaussianNaiveBayes	0.18894	0.01004
	GradientBoostingClassifier	0.04213	0.00515
	K_Neighbours	0.07768	0.00687
	NN-12-layer_wide_with_dropout	0.40355	0.01259
	NN-12-layer_wide_with_dropout_lr01	0.40355	0.01259
	NN-12-layer_wide_with_dropout_lr1	0.40355	0.01259
	NN-2-layer-droput-input-layer_lr001	0.07176	0.00662
	NN-2-layer-droput-input-layer_lr01	0.40355	0.01259
	NN-2-layer-droput-input-layer_lr1	0.40355	0.01259
	NN-4-layer-droput-each-layer_lr0001	0.07176	0.00662

Continued on next page

		loss	std_error
	NN-4-layer-droput-each-layer_lr01	0.40355	0.01259
	NN-4-layer-droput-each-layer_lr1	0.40355	0.01259
	NN-4-layer_thin_dropout	0.07242	0.00665
	NN-4-layer_thin_dropout_lr01	0.40355	0.01259
	NN-4-layer_thin_dropout_lr1	0.40355	0.01259
	NN-4-layer_wide_no_dropout	0.07242	0.00665
	NN-4-layer_wide_no_dropout_lr01	0.40355	0.01259
	NN-4-layer_wide_no_dropout_lr1	0.59645	0.01259
	NN-4-layer_wide_with_dropout	0.07900	0.00692
	NN-4-layer_wide_with_dropout_lr01	0.40355	0.01259
	NN-4-layer_wide_with_dropout_lr1	0.40355	0.01259
	PassiveAggressiveClassifier	0.08097	0.00700
	RandomForestClassifier	0.04411	0.00527
	SVC	0.06649	0.00639
spect	BaggingClassifier	0.43182	0.05280
	BaselineClassifier	0.51136	0.05329
	BernoulliNaiveBayes	0.37500	0.05161
	GaussianNaiveBayes	0.37500	0.05161
	GradientBoostingClassifier	0.37500	0.05161
	K_Neighbours	0.38636	0.05191
	NN-12-layer_wide_with_dropout	0.38636	0.05191
	NN-12-layer_wide_with_dropout_lr01	0.38636	0.05191
	NN-12-layer_wide_with_dropout_lr1	0.38636	0.05191
	NN-2-layer-droput-input-layer_lr001	0.39773	0.05217
	NN-2-layer-droput-input-layer_lr01	0.38636	0.05191
	NN-2-layer-droput-input-layer_lr1	0.38636	0.05191
	NN-4-layer-droput-each-layer_lr0001	0.28409	0.04807
	NN-4-layer-droput-each-layer_lr01	0.38636	0.05191
	NN-4-layer-droput-each-layer_lr1	0.38636	0.05191
	NN-4-layer_thin_dropout	0.34091	0.05053
	NN-4-layer_thin_dropout_lr01	0.38636	0.05191
	NN-4-layer_thin_dropout_lr1	0.61364	0.05191
	NN-4-layer_wide_no_dropout	0.37500	0.05161
	NN-4-layer_wide_no_dropout_lr01	0.61364	0.05191
	NN-4-layer_wide_no_dropout_lr1	0.61364	0.05191
	NN-4-layer_wide_with_dropout	0.47727	0.05325
	NN-4-layer_wide_with_dropout_lr01	0.38636	0.05191
	NN-4-layer_wide_with_dropout_lr1	0.38636	0.05191
	PassiveAggressiveClassifier	0.40909	0.05241
	RandomForestClassifier	0.36364	0.05128
	SVC	0.31818	0.04965
spectf	BaggingClassifier	0.13483	0.03620
	BaselineClassifier	0.38202	0.05150
	BernoulliNaiveBayes	0.28090	0.04764
	GaussianNaiveBayes	0.21348	0.04344
	GradientBoostingClassifier	0.15730	0.03859
	K_Neighbours	0.17978	0.04070
	NN-12-layer_wide_with_dropout	0.20225	0.04258
	NN-12-layer_wide_with_dropout_lr01	0.20225	0.04258
	NN-12-layer_wide_with_dropout_lr1	0.20225	0.04258
	NN-2-layer-droput-input-layer_lr001	0.20225	0.04258
	NN-2-layer-droput-input-layer_lr01	0.20225	0.04258
	NN-2-layer-droput-input-layer_lr1	0.20225	0.04258
	NN-4-layer-droput-each-layer_lr0001	0.20225	0.04258
	NN-4-layer-droput-each-layer_lr01	0.20225	0.04258

Continued on next page

		loss	std_error
	NN-4-layer-droput-each-layer_lr1	0.20225	0.04258
	NN-4-layer_thin_dropout	0.20225	0.04258
	NN-4-layer_thin_dropout_lr01	0.20225	0.04258
	NN-4-layer_thin_dropout_lr1	0.20225	0.04258
	NN-4-layer_wide_no_dropout	0.20225	0.04258
	NN-4-layer_wide_no_dropout_lr01	0.20225	0.04258
	NN-4-layer_wide_no_dropout_lr1	0.20225	0.04258
	NN-4-layer_wide_with_dropout	0.20225	0.04258
	NN-4-layer_wide_with_dropout_lr01	0.20225	0.04258
	NN-4-layer_wide_with_dropout_lr1	0.20225	0.04258
	PassiveAggressiveClassifier	0.21348	0.04344
	RandomForestClassifier	0.13483	0.03620
	SVC	0.12360	0.03489
statlog_australian_credit	BaggingClassifier	0.35088	0.03161
	BaselineClassifier	0.45614	0.03299
	BernoulliNaiveBayes	0.34649	0.03151
	GaussianNaiveBayes	0.37281	0.03202
	GradientBoostingClassifier	0.35088	0.03161
	K_Neighbours	0.31140	0.03067
	NN-12-layer_wide_with_dropout	0.31579	0.03078
	NN-12-layer_wide_with_dropout_lr01	0.31579	0.03078
	NN-12-layer_wide_with_dropout_lr1	0.31579	0.03078
	NN-2-layer-droput-input-layer_lr001	0.31579	0.03078
	NN-2-layer-droput-input-layer_lr01	0.35965	0.03178
	NN-2-layer-droput-input-layer_lr1	0.31579	0.03078
	NN-4-layer-droput-each-layer_lr0001	0.31579	0.03078
	NN-4-layer-droput-each-layer_lr01	0.31579	0.03078
	NN-4-layer-droput-each-layer_lr1	0.31579	0.03078
	NN-4-layer_thin_dropout	0.31579	0.03078
	NN-4-layer_thin_dropout_lr01	0.31579	0.03078
	NN-4-layer_thin_dropout_lr1	0.31579	0.03078
	NN-4-layer_wide_no_dropout	0.32895	0.03112
	NN-4-layer_wide_no_dropout_lr01	0.31579	0.03078
	NN-4-layer_wide_no_dropout_lr1	0.31579	0.03078
	NN-4-layer_wide_with_dropout	0.31579	0.03078
	NN-4-layer_wide_with_dropout_lr01	0.31579	0.03078
	NN-4-layer_wide_with_dropout_lr1	0.31579	0.03078
	PassiveAggressiveClassifier	0.33333	0.03122
	RandomForestClassifier	0.31140	0.03067
	SVC	0.31579	0.03078
statlog_german_credit	BaggingClassifier	0.24848	0.02379
	BaselineClassifier	0.40303	0.02700
	BernoulliNaiveBayes	0.24545	0.02369
	GaussianNaiveBayes	0.26364	0.02425
	GradientBoostingClassifier	0.24545	0.02369
	K_Neighbours	0.25152	0.02388
	NN-12-layer_wide_with_dropout	0.27879	0.02468
	NN-12-layer_wide_with_dropout_lr01	0.27879	0.02468
	NN-12-layer_wide_with_dropout_lr1	0.27879	0.02468
	NN-2-layer-droput-input-layer_lr001	0.25758	0.02407
	NN-2-layer-droput-input-layer_lr01	0.28182	0.02477
	NN-2-layer-droput-input-layer_lr1	0.27879	0.02468
	NN-4-layer-droput-each-layer_lr0001	0.26667	0.02434
	NN-4-layer-droput-each-layer_lr01	0.27879	0.02468
	NN-4-layer-droput-each-layer_lr1	0.27879	0.02468

Continued on next page

		loss	std_error
	NN-4-layer_thin_dropout	0.27879	0.02468
	NN-4-layer_thin_dropout_lr01	0.27879	0.02468
	NN-4-layer_thin_dropout_lr1	0.27879	0.02468
	NN-4-layer_wide_no_dropout	0.23636	0.02339
	NN-4-layer_wide_no_dropout_lr01	0.27879	0.02468
	NN-4-layer_wide_no_dropout_lr1	0.27879	0.02468
	NN-4-layer_wide_with_dropout	0.24848	0.02379
	NN-4-layer_wide_with_dropout_lr01	0.27879	0.02468
	NN-4-layer_wide_with_dropout_lr1	0.27879	0.02468
	PassiveAggressiveClassifier	0.23030	0.02318
	RandomForestClassifier	0.22424	0.02296
	SVC	0.22424	0.02296
statlog_heart	BaggingClassifier	0.23333	0.04458
	BaselineClassifier	0.42222	0.05206
	BernoulliNaiveBayes	0.20000	0.04216
	GaussianNaiveBayes	0.20000	0.04216
	GradientBoostingClassifier	0.25556	0.04598
	K_Neighbours	0.18889	0.04126
	NN-12-layer_wide_with_dropout	0.50000	0.05270
	NN-12-layer_wide_with_dropout_lr01	0.50000	0.05270
	NN-12-layer_wide_with_dropout_lr1	0.50000	0.05270
	NN-2-layer-droput-input-layer_lr001	0.23333	0.04458
	NN-2-layer-droput-input-layer_lr01	0.23333	0.04458
	NN-2-layer-droput-input-layer_lr1	0.50000	0.05270
	NN-4-layer-droput-each-layer_lr0001	0.20000	0.04216
	NN-4-layer-droput-each-layer_lr01	0.50000	0.05270
	NN-4-layer-droput-each-layer_lr1	0.50000	0.05270
	NN-4-layer_thin_dropout	0.24444	0.04530
	NN-4-layer_thin_dropout_lr01	0.25556	0.04598
	NN-4-layer_thin_dropout_lr1	0.50000	0.05270
	NN-4-layer_wide_no_dropout	0.17778	0.04030
	NN-4-layer_wide_no_dropout_lr01	0.50000	0.05270
	NN-4-layer_wide_no_dropout_lr1	0.50000	0.05270
	NN-4-layer_wide_with_dropout	0.18889	0.04126
	NN-4-layer_wide_with_dropout_lr01	0.50000	0.05270
	NN-4-layer_wide_with_dropout_lr1	0.50000	0.05270
	PassiveAggressiveClassifier	0.18889	0.04126
	RandomForestClassifier	0.20000	0.04216
	SVC	0.22222	0.04382
statlog_image	BaggingClassifier	0.02228	0.00534
	BaselineClassifier	0.85452	0.01276
	BernoulliNaiveBayes	0.25688	0.01582
	GaussianNaiveBayes	0.21494	0.01487
	GradientBoostingClassifier	0.02490	0.00564
	K_Neighbours	0.03539	0.00669
	NN-12-layer_wide_with_dropout	0.57929	0.01787
	NN-12-layer_wide_with_dropout_lr01	0.86370	0.01242
	NN-12-layer_wide_with_dropout_lr1	0.86894	0.01222
	NN-2-layer-droput-input-layer_lr001	0.22543	0.01513
	NN-2-layer-droput-input-layer_lr01	0.73919	0.01590
	NN-2-layer-droput-input-layer_lr1	0.86894	0.01222
	NN-4-layer-droput-each-layer_lr0001	0.13630	0.01242
	NN-4-layer-droput-each-layer_lr01	0.86370	0.01242
	NN-4-layer-droput-each-layer_lr1	0.86894	0.01222
	NN-4-layer_thin_dropout	0.20970	0.01474

Continued on next page

		loss	std_error
	NN-4-layer_thin_dropout_lr01	0.84928	0.01295
	NN-4-layer_thin_dropout_lr1	0.84666	0.01304
	NN-4-layer_wide_no_dropout	0.12058	0.01179
	NN-4-layer_wide_no_dropout_lr01	0.84797	0.01300
	NN-4-layer_wide_no_dropout_lr1	0.86894	0.01222
	NN-4-layer_wide_with_dropout	0.11009	0.01133
	NN-4-layer_wide_with_dropout_lr01	0.86370	0.01242
	NN-4-layer_wide_with_dropout_lr1	0.84797	0.01300
	PassiveAggressiveClassifier	0.07995	0.00982
	RandomForestClassifier	0.01966	0.00503
	SVC	0.05242	0.00807
statlog_landsat	BaggingClassifier	0.10169	0.00656
	BaselineClassifier	0.81874	0.00836
	BernoulliNaiveBayes	0.31450	0.01007
	GaussianNaiveBayes	0.21516	0.00892
	GradientBoostingClassifier	0.10829	0.00674
	K_Neighbours	0.09887	0.00648
	NN-12-layer_wide_with_dropout	0.25471	0.00945
	NN-12-layer_wide_with_dropout_lr01	0.78249	0.00895
	NN-12-layer_wide_with_dropout_lr1	0.78249	0.00895
	NN-2-layer-droput-input-layer_lr001	0.18267	0.00838
	NN-2-layer-droput-input-layer_lr01	0.54802	0.01080
	NN-2-layer-droput-input-layer_lr1	0.77260	0.00909
	NN-4-layer-droput-each-layer_lr0001	0.16808	0.00811
	NN-4-layer-droput-each-layer_lr01	0.78814	0.00887
	NN-4-layer-droput-each-layer_lr1	0.88889	0.00682
	NN-4-layer_thin_dropout	0.16384	0.00803
	NN-4-layer_thin_dropout_lr01	0.75047	0.00939
	NN-4-layer_thin_dropout_lr1	0.75047	0.00939
	NN-4-layer_wide_no_dropout	0.25800	0.00949
	NN-4-layer_wide_no_dropout_lr01	0.78814	0.00887
	NN-4-layer_wide_no_dropout_lr1	0.75047	0.00939
	NN-4-layer_wide_with_dropout	0.18409	0.00841
	NN-4-layer_wide_with_dropout_lr01	0.78249	0.00895
	NN-4-layer_wide_with_dropout_lr1	0.78249	0.00895
	PassiveAggressiveClassifier	0.18173	0.00837
	RandomForestClassifier	0.09934	0.00649
	SVC	0.10405	0.00662
statlog_shuttle	BaggingClassifier	0.00005	0.00005
	BaselineClassifier	0.35418	0.00346
	BernoulliNaiveBayes	0.11223	0.00228
	GaussianNaiveBayes	0.16217	0.00266
	GradientBoostingClassifier	0.00005	0.00005
	K_Neighbours	0.00146	0.00028
	NN-12-layer_wide_with_dropout	0.21484	0.00297
	NN-12-layer_wide_with_dropout_lr01	0.21484	0.00297
	NN-12-layer_wide_with_dropout_lr1	0.21484	0.00297
	NN-2-layer-droput-input-layer_lr001	0.33323	0.00341
	NN-2-layer-droput-input-layer_lr01	0.21484	0.00297
	NN-2-layer-droput-input-layer_lr1	0.21484	0.00297
	NN-4-layer-droput-each-layer_lr0001	0.34922	0.00345
	NN-4-layer-droput-each-layer_lr01	0.21484	0.00297
	NN-4-layer-droput-each-layer_lr1	0.21484	0.00297
	NN-4-layer_thin_dropout	0.35648	0.00346
	NN-4-layer_thin_dropout_lr01	0.21484	0.00297

Continued on next page

		loss	std_error
	NN-4-layer_thin_dropout_lr1	0.94227	0.00169
	NN-4-layer_wide_no_dropout	0.34943	0.00345
	NN-4-layer_wide_no_dropout_lr01	0.21484	0.00297
	NN-4-layer_wide_no_dropout_lr1	0.21484	0.00297
	NN-4-layer_wide_with_dropout	0.34775	0.00344
	NN-4-layer_wide_with_dropout_lr01	0.21484	0.00297
	NN-4-layer_wide_with_dropout_lr1	0.21484	0.00297
	PassiveAggressiveClassifier	0.04838	0.00155
	RandomForestClassifier	0.00016	0.00009
	SVC	0.00146	0.00028
statlog_vehicle	BaggingClassifier	0.26071	0.02624
	BaselineClassifier	0.78571	0.02452
	BernoulliNaiveBayes	0.57143	0.02957
	GaussianNaiveBayes	0.52500	0.02984
	GradientBoostingClassifier	0.29643	0.02729
	K_Neighbours	0.30000	0.02739
	NN-12-layer_wide_with_dropout	0.61429	0.02909
	NN-12-layer_wide_with_dropout_lr01	0.75357	0.02575
	NN-12-layer_wide_with_dropout_lr1	0.76429	0.02537
	NN-2-layer-droput-input-layer_lr001	0.45357	0.02975
	NN-2-layer-droput-input-layer_lr01	0.64286	0.02864
	NN-2-layer-droput-input-layer_lr1	0.76429	0.02537
	NN-4-layer-droput-each-layer_lr0001	0.47857	0.02985
	NN-4-layer-droput-each-layer_lr01	0.76429	0.02537
	NN-4-layer-droput-each-layer_lr1	0.76429	0.02537
	NN-4-layer_thin_dropout	0.43571	0.02963
	NN-4-layer_thin_dropout_lr01	0.76429	0.02537
	NN-4-layer_thin_dropout_lr1	0.76429	0.02537
	NN-4-layer_wide_no_dropout	0.27857	0.02679
	NN-4-layer_wide_no_dropout_lr01	0.76071	0.02550
	NN-4-layer_wide_no_dropout_lr1	0.76429	0.02537
	NN-4-layer_wide_with_dropout	0.32143	0.02791
	NN-4-layer_wide_with_dropout_lr01	0.76429	0.02537
	NN-4-layer_wide_with_dropout_lr1	0.75357	0.02575
	PassiveAggressiveClassifier	0.20000	0.02390
	RandomForestClassifier	0.26429	0.02635
	SVC	0.17857	0.02289
steel_plates	BaggingClassifier	0.23089	0.01664
	BaselineClassifier	0.80499	0.01565
	BernoulliNaiveBayes	0.39470	0.01931
	GaussianNaiveBayes	0.46490	0.01970
	GradientBoostingClassifier	0.23089	0.01664
	K_Neighbours	0.26677	0.01747
	NN-12-layer_wide_with_dropout	0.48674	0.01974
	NN-12-layer_wide_with_dropout_lr01	0.80031	0.01579
	NN-12-layer_wide_with_dropout_lr1	0.64587	0.01889
	NN-2-layer-droput-input-layer_lr001	0.35725	0.01893
	NN-2-layer-droput-input-layer_lr01	0.64587	0.01889
	NN-2-layer-droput-input-layer_lr1	0.64587	0.01889
	NN-4-layer-droput-each-layer_lr0001	0.39158	0.01928
	NN-4-layer-droput-each-layer_lr01	0.64587	0.01889
	NN-4-layer-droput-each-layer_lr1	0.64587	0.01889
	NN-4-layer_thin_dropout	0.45554	0.01967
	NN-4-layer_thin_dropout_lr01	0.64587	0.01889
	NN-4-layer_thin_dropout_lr1	0.64587	0.01889

Continued on next page

		loss	std_error
	NN-4-layer_wide_no_dropout	0.31513	0.01835
	NN-4-layer_wide_no_dropout_lr01	0.64587	0.01889
	NN-4-layer_wide_no_dropout_lr1	0.64587	0.01889
	NN-4-layer_wide_with_dropout	0.29641	0.01804
	NN-4-layer_wide_with_dropout_lr01	0.78783	0.01615
	NN-4-layer_wide_with_dropout_lr1	0.64587	0.01889
	PassiveAggressiveClassifier	0.33697	0.01867
	RandomForestClassifier	0.23089	0.01664
	SVC	0.25585	0.01723
synthetic_control	BaggingClassifier	0.02525	0.01115
	BaselineClassifier	0.81313	0.02770
	BernoulliNaiveBayes	0.04545	0.01480
	GaussianNaiveBayes	0.05556	0.01628
	GradientBoostingClassifier	0.06061	0.01696
	K_Neighbours	0.03030	0.01218
	NN-12-layer_wide_with_dropout	0.65657	0.03375
	NN-12-layer_wide_with_dropout_lr01	0.84343	0.02583
	NN-12-layer_wide_with_dropout_lr1	0.85354	0.02513
	NN-2-layer-droput-input-layer_lr001	0.37879	0.03447
	NN-2-layer-droput-input-layer_lr01	0.85354	0.02513
	NN-2-layer-droput-input-layer_lr1	0.84343	0.02583
	NN-4-layer-droput-each-layer_lr0001	0.37374	0.03438
	NN-4-layer-droput-each-layer_lr01	0.85354	0.02513
	NN-4-layer-droput-each-layer_lr1	0.82828	0.02680
	NN-4-layer_thin_dropout	0.52525	0.03549
	NN-4-layer_thin_dropout_lr01	0.82828	0.02680
	NN-4-layer_thin_dropout_lr1	0.82828	0.02680
	NN-4-layer_wide_no_dropout	0.24747	0.03067
	NN-4-layer_wide_no_dropout_lr01	0.82323	0.02711
	NN-4-layer_wide_no_dropout_lr1	0.82828	0.02680
	NN-4-layer_wide_with_dropout	0.18687	0.02770
	NN-4-layer_wide_with_dropout_lr01	0.82828	0.02680
	NN-4-layer_wide_with_dropout_lr1	0.84343	0.02583
	PassiveAggressiveClassifier	0.14646	0.02513
	RandomForestClassifier	0.00505	0.00504
	SVC	0.01010	0.00711
teaching	BaggingClassifier	0.28000	0.06350
	BaselineClassifier	0.66000	0.06699
	BernoulliNaiveBayes	0.46000	0.07048
	GaussianNaiveBayes	0.52000	0.07065
	GradientBoostingClassifier	0.32000	0.06597
	K_Neighbours	0.38000	0.06864
	NN-12-layer_wide_with_dropout	0.64000	0.06788
	NN-12-layer_wide_with_dropout_lr01	0.70000	0.06481
	NN-12-layer_wide_with_dropout_lr1	0.66000	0.06699
	NN-2-layer-droput-input-layer_lr001	0.64000	0.06788
	NN-2-layer-droput-input-layer_lr01	0.66000	0.06699
	NN-2-layer-droput-input-layer_lr1	0.66000	0.06699
	NN-4-layer-droput-each-layer_lr0001	0.62000	0.06864
	NN-4-layer-droput-each-layer_lr01	0.70000	0.06481
	NN-4-layer-droput-each-layer_lr1	0.70000	0.06481
	NN-4-layer_thin_dropout	0.52000	0.07065
	NN-4-layer_thin_dropout_lr01	0.64000	0.06788
	NN-4-layer_thin_dropout_lr1	0.70000	0.06481
	NN-4-layer_wide_no_dropout	0.44000	0.07020

Continued on next page

		loss	std_error
	NN-4-layer_wide_no_dropout_lr01	0.70000	0.06481
	NN-4-layer_wide_no_dropout_lr1	0.70000	0.06481
	NN-4-layer_wide_with_dropout	0.62000	0.06864
	NN-4-layer_wide_with_dropout_lr01	0.64000	0.06788
	NN-4-layer_wide_with_dropout_lr1	0.70000	0.06481
	PassiveAggressiveClassifier	0.44000	0.07020
	RandomForestClassifier	0.36000	0.06788
	SVC	0.42000	0.06980
thyroid	BaggingClassifier	0.00463	0.00139
	BaselineClassifier	0.14604	0.00724
	BernoulliNaiveBayes	0.06019	0.00488
	GaussianNaiveBayes	0.76515	0.00870
	GradientBoostingClassifier	0.00337	0.00119
	K_Neighbours	0.05261	0.00458
	NN-12-layer_wide_with_dropout	0.07828	0.00551
	NN-12-layer_wide_with_dropout_lr01	0.07828	0.00551
	NN-12-layer_wide_with_dropout_lr1	0.97769	0.00303
	NN-2-layer-droput-input-layer_lr001	0.06524	0.00507
	NN-2-layer-droput-input-layer_lr01	0.07828	0.00551
	NN-2-layer-droput-input-layer_lr1	0.07828	0.00551
	NN-4-layer-droput-each-layer_lr0001	0.07828	0.00551
	NN-4-layer-droput-each-layer_lr01	0.07828	0.00551
	NN-4-layer-droput-each-layer_lr1	0.07828	0.00551
	NN-4-layer_thin_dropout	0.06397	0.00502
	NN-4-layer_thin_dropout_lr01	0.07828	0.00551
	NN-4-layer_thin_dropout_lr1	0.07828	0.00551
	NN-4-layer_wide_no_dropout	0.07828	0.00551
	NN-4-layer_wide_no_dropout_lr01	0.07828	0.00551
	NN-4-layer_wide_no_dropout_lr1	0.07828	0.00551
	NN-4-layer_wide_with_dropout	0.07828	0.00551
	NN-4-layer_wide_with_dropout_lr01	0.07828	0.00551
	NN-4-layer_wide_with_dropout_lr1	0.07828	0.00551
	PassiveAggressiveClassifier	0.05598	0.00472
	RandomForestClassifier	0.00463	0.00139
	SVC	0.03662	0.00385
tic_tac_toe	BaggingClassifier	0.02524	0.00881
	BaselineClassifier	0.45110	0.02795
	BernoulliNaiveBayes	0.24290	0.02409
	GaussianNaiveBayes	0.27445	0.02506
	GradientBoostingClassifier	0.05363	0.01265
	K_Neighbours	0.00000	0.00000
	NN-12-layer_wide_with_dropout	0.12934	0.01885
	NN-12-layer_wide_with_dropout_lr01	0.35962	0.02695
	NN-12-layer_wide_with_dropout_lr1	0.35962	0.02695
	NN-2-layer-droput-input-layer_lr001	0.36593	0.02705
	NN-2-layer-droput-input-layer_lr01	0.35962	0.02695
	NN-2-layer-droput-input-layer_lr1	0.36278	0.02700
	NN-4-layer-droput-each-layer_lr0001	0.28076	0.02524
	NN-4-layer-droput-each-layer_lr01	0.35962	0.02695
	NN-4-layer-droput-each-layer_lr1	0.35962	0.02695
	NN-4-layer_thin_dropout	0.30284	0.02581
	NN-4-layer_thin_dropout_lr01	0.35962	0.02695
	NN-4-layer_thin_dropout_lr1	0.35962	0.02695
	NN-4-layer_wide_no_dropout	0.17350	0.02127
	NN-4-layer_wide_no_dropout_lr01	0.35962	0.02695

Continued on next page

		loss	std_error
	NN-4-layer_wide_no_dropout_lr1	0.35962	0.02695
	NN-4-layer_wide_with_dropout	0.19558	0.02228
	NN-4-layer_wide_with_dropout_lr01	0.35962	0.02695
	NN-4-layer_wide_with_dropout_lr1	0.35962	0.02695
	PassiveAggressiveClassifier	0.02524	0.00881
	RandomForestClassifier	0.03155	0.00982
	SVC	0.02524	0.00881
titanic	BaggingClassifier	0.24072	0.01586
	BaselineClassifier	0.41541	0.01828
	BernoulliNaiveBayes	0.24347	0.01592
	GaussianNaiveBayes	0.25585	0.01618
	GradientBoostingClassifier	0.24072	0.01586
	K_Neighbours	0.24072	0.01586
	NN-12-layer_wide_with_dropout	0.25860	0.01624
	NN-12-layer_wide_with_dropout_lr01	0.65199	0.01767
	NN-12-layer_wide_with_dropout_lr1	0.34801	0.01767
	NN-2-layer-droput-input-layer_lr001	0.25860	0.01624
	NN-2-layer-droput-input-layer_lr01	0.46905	0.01851
	NN-2-layer-droput-input-layer_lr1	0.37552	0.01796
	NN-4-layer-droput-each-layer_lr0001	0.25585	0.01618
	NN-4-layer-droput-each-layer_lr01	0.34801	0.01767
	NN-4-layer-droput-each-layer_lr1	0.34801	0.01767
	NN-4-layer_thin_dropout	0.25034	0.01607
	NN-4-layer_thin_dropout_lr01	0.34801	0.01767
	NN-4-layer_thin_dropout_lr1	0.34801	0.01767
	NN-4-layer_wide_no_dropout	0.28061	0.01666
	NN-4-layer_wide_no_dropout_lr01	0.34801	0.01767
	NN-4-layer_wide_no_dropout_lr1	0.34801	0.01767
	NN-4-layer_wide_with_dropout	0.26823	0.01643
	NN-4-layer_wide_with_dropout_lr01	0.65199	0.01767
	NN-4-layer_wide_with_dropout_lr1	0.65199	0.01767
	PassiveAggressiveClassifier	0.25172	0.01610
	RandomForestClassifier	0.24072	0.01586
	SVC	0.24072	0.01586
vertebral_column_2clases	BaggingClassifier	0.15534	0.03569
	BaselineClassifier	0.38835	0.04802
	BernoulliNaiveBayes	0.23301	0.04165
	GaussianNaiveBayes	0.16505	0.03658
	GradientBoostingClassifier	0.20388	0.03970
	K_Neighbours	0.21359	0.04038
	NN-12-layer_wide_with_dropout	0.31068	0.04560
	NN-12-layer_wide_with_dropout_lr01	0.31068	0.04560
	NN-12-layer_wide_with_dropout_lr1	0.31068	0.04560
	NN-2-layer-droput-input-layer_lr001	0.31068	0.04560
	NN-2-layer-droput-input-layer_lr01	0.31068	0.04560
	NN-2-layer-droput-input-layer_lr1	0.31068	0.04560
	NN-4-layer-droput-each-layer_lr0001	0.21359	0.04038
	NN-4-layer-droput-each-layer_lr01	0.31068	0.04560
	NN-4-layer-droput-each-layer_lr1	0.31068	0.04560
	NN-4-layer_thin_dropout	0.20388	0.03970
	NN-4-layer_thin_dropout_lr01	0.31068	0.04560
	NN-4-layer_thin_dropout_lr1	0.31068	0.04560
	NN-4-layer_wide_no_dropout	0.14563	0.03476
	NN-4-layer_wide_no_dropout_lr01	0.31068	0.04560
	NN-4-layer_wide_no_dropout_lr1	0.31068	0.04560

Continued on next page

		loss	std_error
	NN-4-layer_wide_with_dropout	0.19417	0.03898
	NN-4-layer_wide_with_dropout_lr01	0.31068	0.04560
	NN-4-layer_wide_with_dropout_lr1	0.31068	0.04560
	PassiveAggressiveClassifier	0.10680	0.03043
	RandomForestClassifier	0.11650	0.03161
	SVC	0.09709	0.02917
vertebral_column_3clases	BaggingClassifier	0.15534	0.03569
	BaselineClassifier	0.61165	0.04802
	BernoulliNaiveBayes	0.19417	0.03898
	GaussianNaiveBayes	0.17476	0.03742
	GradientBoostingClassifier	0.21359	0.04038
	K_Neighbours	0.20388	0.03970
	NN-12-layer_wide_with_dropout	0.48544	0.04925
	NN-12-layer_wide_with_dropout_lr01	0.48544	0.04925
	NN-12-layer_wide_with_dropout_lr1	0.48544	0.04925
	NN-2-layer-droput-input-layer_lr001	0.28155	0.04432
	NN-2-layer-droput-input-layer_lr01	0.30097	0.04520
	NN-2-layer-droput-input-layer_lr1	0.30097	0.04520
	NN-4-layer-droput-each-layer_lr0001	0.31068	0.04560
	NN-4-layer-droput-each-layer_lr01	0.48544	0.04925
	NN-4-layer-droput-each-layer_lr1	0.48544	0.04925
	NN-4-layer_thin_dropout	0.22330	0.04103
	NN-4-layer_thin_dropout_lr01	0.48544	0.04925
	NN-4-layer_thin_dropout_lr1	0.48544	0.04925
	NN-4-layer_wide_no_dropout	0.21359	0.04038
	NN-4-layer_wide_no_dropout_lr01	0.48544	0.04925
	NN-4-layer_wide_no_dropout_lr1	0.48544	0.04925
	NN-4-layer_wide_with_dropout	0.17476	0.03742
	NN-4-layer_wide_with_dropout_lr01	0.48544	0.04925
	NN-4-layer_wide_with_dropout_lr1	0.48544	0.04925
	PassiveAggressiveClassifier	0.13592	0.03377
	RandomForestClassifier	0.13592	0.03377
	SVC	0.11650	0.03161
wall_following	BaggingClassifier	0.00611	0.00184
	BaselineClassifier	0.65630	0.01119
	BernoulliNaiveBayes	0.38590	0.01147
	GaussianNaiveBayes	0.46807	0.01176
	GradientBoostingClassifier	0.00500	0.00166
	K_Neighbours	0.13215	0.00798
	NN-12-layer_wide_with_dropout	0.29928	0.01079
	NN-12-layer_wide_with_dropout_lr01	0.61188	0.01148
	NN-12-layer_wide_with_dropout_lr1	0.59245	0.01158
	NN-2-layer-droput-input-layer_lr001	0.34203	0.01118
	NN-2-layer-droput-input-layer_lr01	0.61188	0.01148
	NN-2-layer-droput-input-layer_lr1	0.57746	0.01164
	NN-4-layer-droput-each-layer_lr0001	0.23154	0.00994
	NN-4-layer-droput-each-layer_lr01	0.61188	0.01148
	NN-4-layer-droput-each-layer_lr1	0.59245	0.01158
	NN-4-layer_thin_dropout	0.24431	0.01012
	NN-4-layer_thin_dropout_lr01	0.61188	0.01148
	NN-4-layer_thin_dropout_lr1	0.61188	0.01148
	NN-4-layer_wide_no_dropout	0.19100	0.00926
	NN-4-layer_wide_no_dropout_lr01	0.61188	0.01148
	NN-4-layer_wide_no_dropout_lr1	0.61188	0.01148
	NN-4-layer_wide_with_dropout	0.19267	0.00929

Continued on next page

		loss	std_error
	NN-4-layer_wide_with_dropout_lr01	0.61188	0.01148
	NN-4-layer_wide_with_dropout_lr1	0.59245	0.01158
	PassiveAggressiveClassifier	0.30316	0.01083
	RandomForestClassifier	0.00278	0.00124
	SVC	0.18490	0.00915
waveform	BaggingClassifier	0.15818	0.00898
	BaselineClassifier	0.65576	0.01170
	BernoulliNaiveBayes	0.19455	0.00975
	GaussianNaiveBayes	0.17152	0.00928
	GradientBoostingClassifier	0.16182	0.00907
	K_Neighbours	0.14182	0.00859
	NN-12-layer_wide_with_dropout	0.22364	0.01026
	NN-12-layer_wide_with_dropout_lr01	0.67758	0.01151
	NN-12-layer_wide_with_dropout_lr1	0.66606	0.01161
	NN-2-layer-droput-input-layer_lr001	0.14848	0.00875
	NN-2-layer-droput-input-layer_lr01	0.50727	0.01231
	NN-2-layer-droput-input-layer_lr1	0.67758	0.01151
	NN-4-layer-droput-each-layer_lr0001	0.16727	0.00919
	NN-4-layer-droput-each-layer_lr01	0.65636	0.01169
	NN-4-layer-droput-each-layer_lr1	0.66606	0.01161
	NN-4-layer_thin_dropout	0.15879	0.00900
	NN-4-layer_thin_dropout_lr01	0.65636	0.01169
	NN-4-layer_thin_dropout_lr1	0.67758	0.01151
	NN-4-layer_wide_no_dropout	0.17576	0.00937
	NN-4-layer_wide_no_dropout_lr01	0.66606	0.01161
	NN-4-layer_wide_no_dropout_lr1	0.67758	0.01151
	NN-4-layer_wide_with_dropout	0.19818	0.00981
	NN-4-layer_wide_with_dropout_lr01	0.65636	0.01169
	NN-4-layer_wide_with_dropout_lr1	0.65636	0.01169
	PassiveAggressiveClassifier	0.16000	0.00903
	RandomForestClassifier	0.14606	0.00869
	SVC	0.12727	0.00820
waveform_noise	BaggingClassifier	0.17212	0.00929
	BaselineClassifier	0.67879	0.01150
	BernoulliNaiveBayes	0.21091	0.01004
	GaussianNaiveBayes	0.18121	0.00948
	GradientBoostingClassifier	0.16909	0.00923
	K_Neighbours	0.17333	0.00932
	NN-12-layer_wide_with_dropout	0.45697	0.01226
	NN-12-layer_wide_with_dropout_lr01	0.67818	0.01150
	NN-12-layer_wide_with_dropout_lr1	0.65879	0.01167
	NN-2-layer-droput-input-layer_lr001	0.16970	0.00924
	NN-2-layer-droput-input-layer_lr01	0.38242	0.01196
	NN-2-layer-droput-input-layer_lr1	0.58242	0.01214
	NN-4-layer-droput-each-layer_lr0001	0.15636	0.00894
	NN-4-layer-droput-each-layer_lr01	0.67818	0.01150
	NN-4-layer-droput-each-layer_lr1	0.67818	0.01150
	NN-4-layer_thin_dropout	0.16606	0.00916
	NN-4-layer_thin_dropout_lr01	0.67818	0.01150
	NN-4-layer_thin_dropout_lr1	0.65879	0.01167
	NN-4-layer_wide_no_dropout	0.20909	0.01001
	NN-4-layer_wide_no_dropout_lr01	0.65879	0.01167
	NN-4-layer_wide_no_dropout_lr1	0.67818	0.01150
	NN-4-layer_wide_with_dropout	0.22848	0.01034
	NN-4-layer_wide_with_dropout_lr01	0.67818	0.01150

Continued on next page

		loss	std_error
	NN-4-layer_wide_with_dropout_lr1	0.67818	0.01150
	PassiveAggressiveClassifier	0.16485	0.00913
	RandomForestClassifier	0.14667	0.00871
	SVC	0.14000	0.00854
wine	BaggingClassifier	0.03390	0.02356
	BaselineClassifier	0.66102	0.06163
	BernoulliNaiveBayes	0.06780	0.03273
	GaussianNaiveBayes	0.03390	0.02356
	GradientBoostingClassifier	0.03390	0.02356
	K_Neighbours	0.03390	0.02356
	NN-12-layer_wide_with_dropout	0.59322	0.06395
	NN-12-layer_wide_with_dropout_lr01	0.62712	0.06296
	NN-12-layer_wide_with_dropout_lr1	0.62712	0.06296
	NN-2-layer-droput-input-layer_lr001	0.06780	0.03273
	NN-2-layer-droput-input-layer_lr01	0.05085	0.02860
	NN-2-layer-droput-input-layer_lr1	0.23729	0.05539
	NN-4-layer-droput-each-layer_lr0001	0.01695	0.01680
	NN-4-layer-droput-each-layer_lr01	0.62712	0.06296
	NN-4-layer-droput-each-layer_lr1	0.62712	0.06296
	NN-4-layer_thin_dropout	0.05085	0.02860
	NN-4-layer_thin_dropout_lr01	0.76271	0.05539
	NN-4-layer_thin_dropout_lr1	0.77966	0.05396
	NN-4-layer_wide_no_dropout	0.06780	0.03273
	NN-4-layer_wide_no_dropout_lr01	0.62712	0.06296
	NN-4-layer_wide_no_dropout_lr1	0.62712	0.06296
	NN-4-layer_wide_with_dropout	0.03390	0.02356
	NN-4-layer_wide_with_dropout_lr01	0.62712	0.06296
	NN-4-layer_wide_with_dropout_lr1	0.62712	0.06296
	PassiveAggressiveClassifier	0.01695	0.01680
	RandomForestClassifier	0.01695	0.01680
	SVC	0.05085	0.02860
wine_quality_red	BaggingClassifier	0.32386	0.02036
	BaselineClassifier	0.63826	0.02091
	BernoulliNaiveBayes	0.44697	0.02164
	GaussianNaiveBayes	0.44318	0.02162
	GradientBoostingClassifier	0.38068	0.02113
	K_Neighbours	0.33712	0.02057
	NN-12-layer_wide_with_dropout	0.41477	0.02144
	NN-12-layer_wide_with_dropout_lr01	0.57386	0.02152
	NN-12-layer_wide_with_dropout_lr1	0.57386	0.02152
	NN-2-layer-droput-input-layer_lr001	0.43561	0.02158
	NN-2-layer-droput-input-layer_lr01	0.62879	0.02103
	NN-2-layer-droput-input-layer_lr1	0.60227	0.02130
	NN-4-layer-droput-each-layer_lr0001	0.41288	0.02143
	NN-4-layer-droput-each-layer_lr01	0.57386	0.02152
	NN-4-layer-droput-each-layer_lr1	0.60227	0.02130
	NN-4-layer_thin_dropout	0.44508	0.02163
	NN-4-layer_thin_dropout_lr01	0.55492	0.02163
	NN-4-layer_thin_dropout_lr1	0.57386	0.02152
	NN-4-layer_wide_no_dropout	0.40909	0.02140
	NN-4-layer_wide_no_dropout_lr01	0.57386	0.02152
	NN-4-layer_wide_no_dropout_lr1	0.60227	0.02130
	NN-4-layer_wide_with_dropout	0.43561	0.02158
	NN-4-layer_wide_with_dropout_lr01	0.57386	0.02152
	NN-4-layer_wide_with_dropout_lr1	0.60227	0.02130

Continued on next page

		loss	std_error
	PassiveAggressiveClassifier	0.46591	0.02171
	RandomForestClassifier	0.31061	0.02014
	SVC	0.36553	0.02096
wine_quality_white	BaggingClassifier	0.34447	0.01182
	BaselineClassifier	0.65368	0.01183
	BernoulliNaiveBayes	0.53618	0.01240
	GaussianNaiveBayes	0.54298	0.01239
	GradientBoostingClassifier	0.34694	0.01184
	K_Neighbours	0.33457	0.01173
	NN-12-layer_wide_with_dropout	0.55226	0.01237
	NN-12-layer_wide_with_dropout_lr01	0.56277	0.01234
	NN-12-layer_wide_with_dropout_lr1	0.56277	0.01234
	NN-2-layer-droput-input-layer_lr001	0.51082	0.01243
	NN-2-layer-droput-input-layer_lr01	0.56277	0.01234
	NN-2-layer-droput-input-layer_lr1	0.56277	0.01234
	NN-4-layer-droput-each-layer_lr0001	0.49722	0.01243
	NN-4-layer-droput-each-layer_lr01	0.56277	0.01234
	NN-4-layer-droput-each-layer_lr1	0.56277	0.01234
	NN-4-layer_thin_dropout	0.46753	0.01241
	NN-4-layer_thin_dropout_lr01	0.56277	0.01234
	NN-4-layer_thin_dropout_lr1	0.56277	0.01234
	NN-4-layer_wide_no_dropout	0.49845	0.01243
	NN-4-layer_wide_no_dropout_lr01	0.56277	0.01234
	NN-4-layer_wide_no_dropout_lr1	0.56277	0.01234
	NN-4-layer_wide_with_dropout	0.49103	0.01243
	NN-4-layer_wide_with_dropout_lr01	0.56277	0.01234
	NN-4-layer_wide_with_dropout_lr1	0.69759	0.01142
	PassiveAggressiveClassifier	0.50959	0.01243
	RandomForestClassifier	0.33952	0.01178
	SVC	0.35498	0.01190
yeast	BaggingClassifier	0.42449	0.02233
	BaselineClassifier	0.75918	0.01932
	BernoulliNaiveBayes	0.52449	0.02256
	GaussianNaiveBayes	0.85918	0.01571
	GradientBoostingClassifier	0.44490	0.02245
	K_Neighbours	0.44286	0.02244
	NN-12-layer_wide_with_dropout	0.71224	0.02045
	NN-12-layer_wide_with_dropout_lr01	0.70204	0.02066
	NN-12-layer_wide_with_dropout_lr1	0.71224	0.02045
	NN-2-layer-droput-input-layer_lr001	0.50408	0.02259
	NN-2-layer-droput-input-layer_lr01	0.72245	0.02023
	NN-2-layer-droput-input-layer_lr1	0.64286	0.02165
	NN-4-layer-droput-each-layer_lr0001	0.50408	0.02259
	NN-4-layer-droput-each-layer_lr01	0.71224	0.02045
	NN-4-layer-droput-each-layer_lr1	0.70204	0.02066
	NN-4-layer_thin_dropout	0.50000	0.02259
	NN-4-layer_thin_dropout_lr01	0.70204	0.02066
	NN-4-layer_thin_dropout_lr1	0.70204	0.02066
	NN-4-layer_wide_no_dropout	0.50408	0.02259
	NN-4-layer_wide_no_dropout_lr01	0.71224	0.02045
	NN-4-layer_wide_no_dropout_lr1	0.71224	0.02045
	NN-4-layer_wide_with_dropout	0.48163	0.02257
	NN-4-layer_wide_with_dropout_lr01	0.70204	0.02066
	NN-4-layer_wide_with_dropout_lr1	0.70204	0.02066
	PassiveAggressiveClassifier	0.47347	0.02256

Continued on next page

		loss	std_error
	RandomForestClassifier	0.40816	0.02220
	SVC	0.44286	0.02244
zoo	BaggingClassifier	0.02941	0.02898
	BaselineClassifier	0.85294	0.06074
	BernoulliNaiveBayes	0.02941	0.02898
	GaussianNaiveBayes	0.02941	0.02898
	GradientBoostingClassifier	0.08824	0.04864
	K_Neighbours	0.02941	0.02898
	NN-12-layer_wide_with_dropout	0.88235	0.05526
	NN-12-layer_wide_with_dropout_lr01	0.50000	0.08575
	NN-12-layer_wide_with_dropout_lr1	0.85294	0.06074
	NN-2-layer-droput-input-layer_lr001	0.23529	0.07275
	NN-2-layer-droput-input-layer_lr01	0.44118	0.08515
	NN-2-layer-droput-input-layer_lr1	0.50000	0.08575
	NN-4-layer-droput-each-layer_lr0001	0.23529	0.07275
	NN-4-layer-droput-each-layer_lr01	0.50000	0.08575
	NN-4-layer-droput-each-layer_lr1	0.85294	0.06074
	NN-4-layer_thin_dropout	0.17647	0.06538
	NN-4-layer_thin_dropout_lr01	0.91176	0.04864
	NN-4-layer_thin_dropout_lr1	0.88235	0.05526
	NN-4-layer_wide_no_dropout	0.11765	0.05526
	NN-4-layer_wide_no_dropout_lr01	0.50000	0.08575
	NN-4-layer_wide_no_dropout_lr1	0.50000	0.08575
	NN-4-layer_wide_with_dropout	0.14706	0.06074
	NN-4-layer_wide_with_dropout_lr01	0.50000	0.08575
	NN-4-layer_wide_with_dropout_lr1	0.50000	0.08575
	PassiveAggressiveClassifier	0.02941	0.02898
	RandomForestClassifier	0.05882	0.04035
	SVC	0.02941	0.02898