An Easy to Use Repository for Comparing and Improving Machine Learning Algorithm Usage

Michael R. Smith[1] and **Andrew White**[2] and **Christophe Giraud-Carrier**[3] and **Tony Martinez**[4]

Abstract. The results from most machine learning experiments are used for a specific purpose and then discarded. This causes significant loss of information and requires rerunning experiments to compare learning algorithms. Often, this also requires a researcher or practitioner to implement another algorithm for comparison, that may not always be correctly implemented. By storing the results from previous experiments, machine learning algorithms can be compared easily and the knowledge gained from them can be used to improve the performance of future machine learning experiments. The purpose of this work is to provide easy access to previous experimental results for learning and comparison. These stored results are comprehensive – storing the prediction for each test instance as well as the learning algorithm, hyperparameters, and training set that were used in the experiment. Previous experimental results are particularly important for meta-learning, which, in a broad sense, is the process of learning from previous machine learning results such that the learning process is improved. While other experiment databases do exist, one of our focuses is on easy access to the data, eliminating any learning curve required to acquire the desired information. We provide meta-learning data sets that are ready to be downloaded for meta-learning experiments. Easy access to previous experimental results aids other researchers looking to do meta-learning and helps in comparing meta-learning algorithms. In addition, simple queries to the underlying database can be made if specific information is desired. We also differ from previous experiment databases in that our database is designed at the instance level, where an instance is an example in a data set. We store the predictions of a learning algorithm trained on a specific training set for each instance in the test set. Data set level information can then be obtained by aggregating the results from the instances. The instance level information can be used for many tasks such as determining the diversity of a classifier or algorithmically determining the optimal subset of training instances for a learning algorithm.

1 Introduction

The quality of an induced model is dependent on, among other aspects, the learning algorithm that is chosen, the hyper-parameter settings for the chosen learning algorithm, and the quality of the training set. Choosing a learning algorithm for a given task, setting its hyper-parameters, and selecting which instances to train on, however, is non-trivial. Meta-learning deals with the problem of how to select a learning algorithm and set its hyper-parameters based on previous experience (results from previous machine learning experiments). Although some research from the machine learning community has focused on meta-learning (e.g., see [17, 5, 2, 3, 8]), much of the focus of machine learning research has been on developing more learning algorithms and/or applying machine learning in specific domains.

Part of the difficulty of meta-learning is due to the lack of accessible results. As meta-learning requires running several learning algorithms and hyperparameter settings over many data sets, gathering results requires large amounts of computational resources. In addition to the computational requirements, results from the learning algorithms may differ due to slight differences in their implementations. Thus, comparing results among meta-learning studies becomes difficult.

To aid in further research in meta-learning, we have developed the *machine learning results repository* (MLRR) that provides data sets ready for download for meta-learning problems, akin to the UCI data repository for machine learning problems. We refer to the data sets for meta-learning as *meta-data sets* to distinguish them from the data sets that are used in the machine learning experiments. The meta-data sets provide a snapshot of an underlying database that stores the results of machine learning experiments. Users can update the database with new results from machine learning experiments and then update the meta-data sets for meta-learning. A revision history is kept so that comparisons among meta-learning algorithms is facilitated. As a starting point, meta-data sets are provided by MLRR for typical meta-learning tasks, such as, given a set of meta-features, predict which learning algorithm and/or hyperparameter setting to use.

The MLRR stores instance level meta-features and the predictions made on each instance by the learning algorithms. Providing information at the instance level allows studies to be performed on the instances themselves. Studying the effects of machine learning on a single instance and/or the effects of a single instance on the performance of an algorithm has generally been overlooked. Instance-level information is important in several areas of machine learning, however. In ensembles, computing the classifier diversity of the ensembled classifiers using the predictions for each instance is important in determining the effectiveness of the ensembling technique [12, 6, 1]. In curriculum learning, the training set is incrementally augmented such that "easier" instances are presented to the learning algorithm first, thus creating a need to understand and identify the easier instances [4]. Smith et al. used instance-level predictions to identify and characterize instances that are likely to be misclassified [23] and used this information to create a curriculum [22]. Other work has also used the instance-level predictions for meta-learning. The classifier output difference (COD) measures the distance between two learning algorithms as the probability that the learning algorithms make different predictions on test instances [16]. Unsupervised meta-

[1] Brigham Young University, USA, email: msmith@axon.cs.byu.edu
[2] Brigham Young University, USA, email: andrewkvavlewhite@gmail.com
[3] Brigham Young University, USA, email: cgc@cs.byu.edu
[4] Brigham Young University, USA, email: martinez@cs.byu.edu

learning (UML) clusters learning algorithms based on their COD scores (rather than accuracy) to examine the behavior of the learning algorithms [13]. Meta-learning for algorithm selection can then be done over the clusters rather than a larger set of learning algorithms to recommend a cluster of learning algorithms that all behave similarly [14]. Additionally, several techniques treat instances individually during the training process, such as filtering instances from the training set based on their instance-level meta-features [21] or weighting the instances [18].

Other attempts have been made at creating a repository for machine learning experiments from which learning can be conducted [20, 24]. However, we feel that they lack simplicity and/or extensibility. In addition to providing instance-level information, we hope to bridge this gap with the MLRR. Probably the most prominent and well-developed data repository is ExpDB, an experiment database that provides a framework for reporting experimental results and their associated workflow [24]. The purpose of ExpDB is to comprehensively store the workflow process of all experiments for reproducibility. One of the results of storing the experiments is that the results can be used for meta-learning. Unfortunately, there is a relatively steep learning curve to access the data due to the inherent complexity involved in storing all of the details about exact reproducibility. Because of this complexity and formality, it is difficult to directly access the information that would be most beneficial for meta-learning, which may deter some potential users. Additionally, ExpDB does not currently support storage and manipulation of any instance level features.

We acknowledge that maintaining a database of previous experiments is not a trivial problem. We do, however, add our voice to support the importance of maintaining a repository of machine learning results and offer an effective solution for storing results from previous experiments. Our primary goal is to maintain simplicity and provide easily accessible data for meta-learning to 1) help promote more research in meta-learning, 2) provide a standard set of data sets for meta-learning algorithm comparison, and 3) continue to stimulate research at the instance level.

We next describe our approach for providing a repository for machine learning meta-data that emphasizes ease of access to the meta-data. MLRR currently has the results from 72 data sets, 9 learning algorithms and 10 hyperparameter settings for each learning algorithm. The database description is provided in Section 3. How to add new experimental results to the database is detailed in Section 4. We then give a more detailed description of the data set level and instance level meta-features that are used in the MLRR. Conclusions and directions for future work are provided in Section 6.

2 Meta-data Set Descriptions

The purpose of the *machine learning results repository* (MLRR) is to provide easy access to the results of previous machine learning experiments for meta-learning at the data set and instance levels. This, in turn, would allow other researchers interested in meta-learning and in better understanding machine learning algorithms direct access to prior results without having to re-run all of the algorithms or learn how to navigate a more complex experiment database. The quality of an induced model for a task is dependent on at least three things:

1. the learning algorithm chosen to induce the model,
2. the hyperparameter settings for the chosen learning algorithm, and
3. the instances used for training.

When we refer to an experiment, we mean the results from training a learning algorithm l with hyperparamter settings λ on a training set t. We first describe how we manage experiment information, and then describe the provided meta-data sets.

2.1 Experiment Information

The information about each experiment is provided in three tables in MLRR. Which learning algorithm and hyperparameters were used is provided in a file structured as shown in Table 1. It provides the toolkit including the version number that was ran, the learning algorithm, and the hyperparameters that were used. This allows for multiple learning algorithms, hyperparameters, and toolkits to be compared. In the examples in Table 1, the class names from the Weka machine learning toolkit [9] and the Waffles machine learning toolkit [7] are shown. LA_seed corresponds to the learning algorithm that was used (LA) and to a seed that represents which hyperparameter setting was used (seed). The LA_seed will be used in other tables as a foreign key to map back to this table. A seed of -1 represents the default hyperparameter settings as many studies examine the default behavior as given in a toolkit and the default parameters are commonly used in practice.

Table 1. The structure of the meta-data set that describes the hyperparameter settings for the learning algorithms stored in the database.

LA_S	Toolkit	Version	Hyperparameters
BP_1	weka	3.6.11	weka.classifiers.functions.MultilayerPerceptron\ – -L 0.261703 -M 0.161703 -H 12 -D
BP_2	weka	3.6.11	weka.classifiers.functions.MultilayerPerceptron\ – -L 0.25807 -M 0.15807 -H 4
BP_3	waffles	13-12-09	neuralnet -addlayer 8 -learningrate 0.1 \ -momentum 0 -windowsepochs 50
⋮	⋮	⋮	⋮
C4.5_1	weka	3.6.11	weka.classifiers.trees.J48 – -C 0.443973 -M 1
⋮	⋮	⋮	⋮

As the parameter values differ between toolkits, there is a mapping provided to distinguish hyperparameter settings. For example, Weka uses the "-L" parameter to set the learning rate in backpropagation while the Waffles toolkit uses "-learningrate". Also, some toolkits have hyperparameters that other implementations of the same learning algorithm do not include. In such cases, an unknown value will be provided in the meta-data set. This mapping is shown in Table 2 for the backpropagation learning algorithm. The first row contains the values used by MLRR. The following rows contain the command-line parameter supplied to a specific toolkit to set that hyperparameter.

Table 2. The structure of the table for mapping learning algorithm hyperparameters between different toolkits for the backpropagation learning algorithm.

	Command line parameters				
toolkit	LR	Mo	HN	DC	WE
weka	-L	-M	-H	-D	?
waffles	-learningrate	-momentum	-addlayer	?	-windowsepochs
⋮	⋮	⋮	⋮	⋮	⋮

A mapping of which instances are used for training is also provided in a separate file. The structure of this table is shown in Table

3. Each row represents an experiment as toolkit_seed_numFolds_fold. The toolkit represents which toolkit was used, the seed represents the random seed that was provided to the toolkit, numFolds represents how many folds were ran, and fold represents in which fold an instance was included for testing. The values in the following columns represent if an instance was used for training or testing. There is one column for each instance in the data set. They are stored as real values. This allows for the situations when training instances have associated weights. In the file, an unknown value of "?" represents a testing instance, otherwise a real value represents a training instance. A value of 0 represents a filtered instance, a value of 1 represents an unweighted training instance and any value between 0 and 1 represents the weight for that training instance. In the cases where there are specific training and testing sets, then the row will be labeled as toolkit_0_0_1 and information for the training set can be entered as before. A random test/training split of the data is represented as toolkit_seed_percentSplit_1 where "percentSplit" represents the percentage of the data set that was used for testing as generated by the toolkit.

Table 3. The structure of the meta-data set that indicates which instances were used for training given a random seed.

toolkit_seed_# folds_fold	1	2	3	...
weka_1_10_1	1	1	1	...
weka_1_10_2	1	0	1	...
⋮	⋮	⋮	⋮	
weka_1_10_10	0.74	1	?	...
weka_2_1_10	?	1	1	...
⋮	⋮	⋮	⋮	

2.2 Meta-data sets

One of the features of MLRR is its focus on storing and presenting instance level information, namely, instance level characteristics and associated predictions from previous experiments. Indeed, the MLRR is designed intentionally from the instance level perspective, from which data set level information can be computed (e.g., accuracy or precision).

As one of the purposes of the MLRR is ease of access, the MLRR stores several data sets in attribute-relation file format (ARFF) which is supported by many machine learning toolkits. In essence, ARFF is a comma or space separated file with attribute information and possible comments. The precomputed meta-data sets include instance level meta-data sets and data set level meta-data sets.

At the instance level, MLRR provides for each data set a meta-data set that stores the instance level meta-features and the prediction from each experiment. This allows for analyses to be done exploring the effects of hyperparameters and learning algorithms at the instance-level, which is currently mostly overlooked. For each data set, a meta-data set is provided that gives the values for the instance level meta-features, the actual class value (stored as a numeric value), and the predicted class value for each experiment. The training set and learning algorithm/hyperparameter information is stored in the column heading as "LA_seed/hyperparameter" where LA is a learning algorithm and hyperparameter is the hyperparameter setting for the learning algorithm. Together, they map to the entries in Table 1. The seed represents the seed that was used to partition the data (see Table 3). The structure of the instance level meta-data set is shown in Table 4. In the given example, instance 77 is shown. The "inst meta" section provides the instance level meta-features for that instance. The actual class label is 2. The predictions from the experiments on this data set are provided in the following columns (i.e., experiment BP_1/1 predicted class 3, BP_N/1 predicted class 2, etc.).

Table 4. The structure of the meta-data set at the instance level.

	inst meta				predictions								
#	*k*AN	MV	...	act	BP_1/1	...	BP_N/1	...	BP_N/M		C4.5_1/1		...
77	0.92	0	...	2	3	...	2	...	2		3		...
⋮	⋮	⋮		⋮	⋮		⋮		⋮		⋮		

At the data set level, several meta-data sets are provided:

- a general meta-data set that stores the data set meta-features and the average N by 10-fold cross-validation accuracy for all of the data sets from a learning algorithm with a given hyperparameter setting.
- for each learning algorithm a meta-data set that stores the data set meta-features, the learning algorithm hyperparameter settings, and the average N by 10-fold cross-validation accuracy for all of the data sets for the given hyperparameter setting.

The structure for the general meta-data set is provided in Table 5. The structure and information of this meta-data set is typical of that used in previous meta-learning studies that provides a mapping from data set meta-features to accuracies obtained by a set of learning algorithms. Most previous studies have been limited to only using the default hyperparameters, however. The MLRR includes the accuracies from multiple hyperparameter settings. The hyperparameter settings from each learning algorithm are denoted by a "LA_#" where LA refers to a learning algorithm and # refers to which hyperparameter setting was used for that learning algorithm.

Table 5. The structure of the meta-data set at the data set level.

	data set meta-features			LA accuracies					
data set	numInst	numAttr	...	BP_1	BP_2	...	BP_N	C4.5_1	...
iris	150	4	...	96.80	95.07	...	93.47	95.60	...
abalone	4177	8	...	20.27	29.84	...	21.91	23.24	...
⋮	⋮	⋮		⋮	⋮	⋮	⋮	⋮	

The meta-data sets for each learning algorithm are designed to aid in algorithmic hyperparameter estimation, i.e., given a data set, can we predict which hyperparameter setting will give the highest classification accuracy. For each learning algorithm, a meta-data set is provided that contains the data set meta-features, the toolkit that was used, the hyperparameter setting and the average accuracy for each unique tool kit/hyperparameter combination. The structure of the meta-data set for each learning algorithm is provided in Table 6. The accuracy ("acc") represents the average accuracy for all k-fold validation runs (i.e., multiple runs of the same learning algorithm with different random seeds to partition the folds). The toolkit is also provided to allow a user to compare toolkits or only do hyperparameter estimation for a single toolkit.

MLRR provides easy access for researchers and practitioners to a large and varying set of meta-data information as shown in the tables above. The provided meta-data sets are a snapshot of an underlying

Table 6. The structure of the table for mapping learning algorithm hyperparameters among toolkits.

	DS meta features			toolkit	hyperparameters			
data set	numInst	numAttr	...	weka	LR	Mo	...	acc
iris	150	4	...	weka	0.71	0.61	...	96.80
iris	150	4	...	weka	0.11	0.25	...	97.04
⋮	⋮	⋮		⋮	⋮	⋮		⋮

database that stores all of the previous experimental results that can be updated as more results are obtained. A revision history of the data sets is provided so that results can be compared even if the meta-data set has been updated.

3 Database Description

MLRR uses MongoDB as the database to store the results from machine learning experiments. MongoDB is a NoSQL database that allows for adding new features (such as new learning algorithms and/hyperparameters), thus, escaping the rigidity of the more traditional SQL databases. This allows for easily expanding the database with new learning algorithms and/or hyperparameters. Of course, this is theoretically also possible in a relational database, provided the database has been designed adequately. For example, one could certainly have, and that would indeed be following good design principles, one table for the algorithms and one table for the hyper parameters with appropriate foreign keys. However, such design requires some amount of foresight. In traditional relational databases, the information that needs to be stored (and how) has to be planned for in advance. Otherwise, when new features are desired, a new schema needs to be created and then the database has to be migrated over to the new schema. With a NoSQL database, new learning algorithms/hyperparameters and other pieces of information can easily be added into the MLRR.

The data is stored as a document database as collections of key-value pairs. Each collection represents the experimental results on a particular data set. In each collection, the keys are LA_hyperparameterSetting. The value then is a JSON text document that stores the results of an experiment (e.g., the results of 10-fold cross-validation on the iris data set using C4.5). These documents also contain pointers to other documents that hold information about training/testing sets for each experiment. The data set/instance level meta-features are stored in separate documents in their respective data set collection. A separate collection stores information about the learning algorithms and their hyperparameters.

The best way to visualize the database is as a hierarchy of key-value pairs as shown in Figure 1. At the top-level, there are collections - these are the individual data sets in the database. Each of them holds a collection of documents that represent an output file, or experiment, named by its learning algorithm with two numbers that correspond to the random seed used to partition the data and the hyperparameter setting. In these documents, the predictions for each instance is stored. Collections for which instances were used for training hyperparameter settings are also included.

4 Extending the Database

The data provided by MLRR only contains a snapshot of current machine learning results. To allow more machine learning results to be added and to allow the MLRR to evolve as the state of machine learning evolves, MLRR provides a method to upload new machine learning results. The MLRR also stores the original data sets to allow a user to add results from additional experiments on the current set of data sets. The results from experimentation on a new data set require that the new data set be uploaded as well as the experimental results. Scripts are provided to calculate the meta-features for the new data set. In the case where a data set is proprietary or has other privacy/licensing issues that prevent it from being posted, the meta-features can be calculated on the data set without storing the actual data set.

Currently, scripts are provided to upload the output from running Weka. This provides a simple way to upload experimental results from a commonly used toolkit. The file is slightly modified such that the first line provides which learning algorithm and hyperparameters were used. The database will have the ability to upload files generated by other toolkits in the future.

Of course, there are issues of data reliability. Currently, all of the results stored in the MLRR are from our experiments. To help with data reliability, we require that the script(s) and executable(s) required to reproduce the results are uploaded along with the results. This allows the results to be verified if their validity is questioned. If the results from an experiment are thought to be invalid, they can can be flagged, and inspected for possible removal from the MLRR.

5 Included Meta-features

In this section, we detail the meta-features that are included in the machine learning results repository (MLRR). We store a set of data set meta-features that have been commonly used in previous meta-learning studies. Specifically, we used the meta-features from Brazdil et al. [5], Ho and Basu [10], Pfahringer et al. [17], and Smith et al. [23]. As the underlying database is a NoSQL database, additional meta-features can be easily added in the future. We now describe the meta-features from each study.

The study by Brazdil et al. [5] examined ranking learning algorithms using instance-based learning. The meta-features are designed to be quickly calculated and to represent properties that affect algorithm performance.

- *Number of examples*. This feature helps identify how scalable an algorithm is based on the size of its input.
- *Proportion of symbolic attributes*. This feature can be used to consider how well an algorithm deals with symbolic or numeric attributes.
- *Proportion of missing values*. This features can be used to consider how robust an algorithm is to incomplete data.
- *Proportion of attributes with outliers*. An attribute is considered to have an outlier if the ratio of variances of the mean value and the α-trimmed mean is smaller than 0.7 where $\alpha = 0.05$. This feature can be used to consider how robust an algorithm is to outlying numeric values.
- *Entropy of classes*. This feature measures one aspect of problem difficulty in the form of whether one class outnumbers another.

Ho and Basu [10] sought to measure the complexity of a data set to identify areas of the data set that contribute to its complexity focusing on the geometrical complexity of the class boundary.

- Measures of overlap of individual feature values:
  - *The maximum Fisher's Discriminant ratio*. This is the Fisher's discriminant ratio for an attribute:

$$f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2},$$

where μ_i and σ_i^2 represent the mean and variance for a class. The maximum Fisher's discriminant value over the attributes is used for this measure. For multiple classes, this measure is expanded to:

$$f = \frac{\sum_{i=1}^{C} \sum_{j=i+1}^{C} p_i p_j (\mu_i - \mu_j)^2}{\sum_{i=1}^{C} p_i \sigma_i^2}$$

where C is the number of classes and p_i is the proportion of instances that belong to the i^{th} class.

- *The overlap of the per-class bounding boxes*. This feature measures the overlap of the tails of the two class-conditional distributions. For data sets with more than 2 classes, the overlap of the per-class bounding boxes is computed for each pair of classes and the sum over all pairs of classes is returned.
- *The maximum (individual) feature efficiency*. This feature measures how discriminative a single feature is. For each attribute, the ratio of instances with differing classes that are not in the overlapping region is returned. The attribute that produces the largest ratio of instances is returned.
- *The collective feature efficiency*. This measure builds off of the previous one. The maximum ratio is first calculated as before. Then, the instances that can be discriminated are removed and the maximum (individual) feature efficiency is recalculated with the remaining instances. This process is repeated until no more instances can be removed. The ratio of instances that can be discriminated is returned.

- Measures of class separability:
  - *The minimized sum of the error distance of a linear classifier*. This feature measures to what extent training data is linearly separable and returns the difference between a linear classifier and the actual class value.
  - *The training error of a linear classifier*. This feature also measures to what extent the training data is linearly separable.
  - *The fraction of points on the class boundary*. This feature estimates the length of the class boundary by constructing a minimum spanning tree over the entire data set and returning the ratio of the number of nodes in the spanning tree that are connected and belong to different classes to the number of instances in the data set.
  - *The ratio of average intra/inter class nearest neighbor distance*. This measure compares the within class spread with the distances to the nearest neighbors of the other classes. For each instance, the distance to its nearest neighbor with the same class ($intraDist(x)$) and to its nearest neighbor with a different class ($interDist(x)$) is calculated. Then the measure returns:

$$\frac{\sum_i^N intraDist(x_i)}{\sum_i^N interDist(x_i)}$$

    where N is the number of instances in the data set.
  - *The leave-one-out error rate of the one-nearest neighbor classifier*. This feature measures how close the examples of different classes are.
- Measures of geometry, topology, and density of manifolds
  - *The nonlinearity of a linear classifier*. Following Hoekstra and Duin [11], given a training set, a test set is created by linear interpolation with random coefficients between pairs of randomly selected instances of the same class. The error rate of a linear classifier trained with the original training set on the generated test set is returned.
  - *The nonlinearity of the one-nearest neighbor classifier*. A test set is created as with the previous feature, but the error rate of a 1-nearest neighbor classifier is returned.
  - *The fraction of maximum covering spheres*. A covering sphere is created by centering on an instance and growing as much as possible before touching an instance from another class. Only the largest spheres are considered. The measure returns the number of spheres divided by the number of instances in the data set and provides an indication of how much the instances are clustered in hyperspheres or distributed in thinner structures.
  - *The average number of points per dimension*. This measure is the ratio of instances to attributes and roughly indicates how sparse a data set is.

Multi-class modifications are made according to the implementation of the data complexity library (DCoL) [15].

Pfahringer et al. [17] introduced the notion of using performance values (i.e., accuracy) of simple and fast classification algorithms as meta-features. The landmarkers that are included in the MLRR are listed below.

- *Linear discriminant learner*. Creates a linear classifier that finds a linear combination of the features to separate the classes.
- *One nearest neighbor learner*. Redundant with the leave-one-out error rate of the one-nearest neighbor classifier from Ho and Basu [10].
- *Decision node learning*. A decision stump that splits on the attribute that has the highest information gain. A decision stump is a decision tree with only one node.
- *Randomly chosen node learner*. A decision stump that splits on a randomly chosen attribute.
- *Worst node learner*. A decision stump that splits on the attribute that has the lowest information gain.
- *Average node learner*. A decision stump is created for each attribute and the average accuracy is returned.

The use of landmarkers has been shown to be competitive with the best performing meta-features with a significant decrease in computational effort [19].

Smith et al. [23] sought to identify and characterize instances that are difficult to classify correctly. The difficulty of an instance was determined based on how frequently it was misclassified. To characterize why some instances are more difficult than others to classify correctly, the authors used different hardness measures. They include:

- *k-Disagreeing Neighbors*. The percentage of k nearest neighbors that do not share the target class of an instance. This measures the local overlap of an instance in the original space of the task.
- *Disjunct size*. This feature indicates how tightly a learning algorithm has to divide the task space to correctly classify an instance. It is measured as the size of a disjunct that covers an instance divided by the largest disjunct produced, where the disjuncts are formed using the C4.5 learning algorithm.
- *Disjunct class percentage*. This features measure the overlap of an instance on a subset of the features. Using a pruned C4.5 tree, the disjunct class percentage is the number of instances in a dis-

junct that belong to the same class divided by the total number of instances in the disjunct.

- *Tree depth (pruned and unpruned)*. Tree depth provides a way to estimate the description length, or Kolmogorov complexity, of an instance. It is the depth of the leaf node that classifies an instance in an induced tree.
- *Class likelihood*. This features provides a global measure of overlap and the likelihood of an instance belonging to the target class. It is calculated as:

$$\prod_{i}^{|x|} p(x_i|t(x))$$

where $|x|$ represents the number of attributes for the instance x and $t(x)$ is the target class of x.

- *Minority value*. This feature measures the skewness of the class that an instance belongs to. It is measured as the ratio of instances sharing the target class of an instance to the number of instances in the majority class.
- *Class balance*. This feature also measures the class skew. First, the ratio of the number of instances belonging the target class to the total number of instances is calculated. The difference of this ratio with the ratio of one over the number of possible classes is returned. If the class were completely balanced (i.e. all class had the same number of instances), a value of 0 would be returned for each instance.

The hardness measures are designed to capture the characteristics of why instances are hard to classify correctly. Data set measures can be generated by averaging the hardness measures over the instances in a data set.

6 Conclusions and Future Work

In this paper, we presented the *machine learning results repository* (MLRR) an easily accessible and extensible database for meta-learning. MLRR was designed with the main goals of providing an easily accessible data repository to facilitate meta-learning and providing benchmark meta-data sets to compare meta-learning experiments. To this end, the MLRR provides ready to download meta-data sets of previous experimental results. One of the important features of MLRR is that it provides meta-data at the instance level. Of course, the results could also be used as a means of comparing one's work with prior work as they are stored in the MLRR. The MLRR can be accessed at `http://axon.cs.byu.edu/mlrr`.

The MLRR allows for reproducible results as the data sets are stored on the server and as the class names and toolkits are provided. The ExpDB tends to be a lot more rigid in its design as it is based on relational databases and PMML (predictive model markup language), thus exhibiting a relatively steep learning curve to import and extract data. The MLRR is less rigid in its design allowing for easier access to the data and more extensibility, with the trade-off of less formality.

One direction for future work is to integrate the API provided at OpenML[5] (an implementation of an experiment database) to incorporate their results with those that are in the MLRR. This will help provide easy access to the results that are already stored in OpenML without having to incur the learning cost associated with understanding the database schema.

Another open problem is how to store information about how a data set is preprocessed. Currently, the MLRR can store the instance level information resulting from preprocessing, but it lacks a mechanism to store the preprocessing process. Integrating this information in an efficient way is a direction of current research.

[5] `www.openml.org`

REFERENCES

[1] M. Aksela and J. Laaksonen, 'Using diversity of errors for selecting members of a committee classifier', *Pattern Recognition*, **39**(4), 608–623, (2006).

[2] S. Ali and K.A. Smith, 'On Learning Algorithm Selection for Classification', *Applied Soft Computing*, **6**2, 119–138, (2006).

[3] S. Ali and K.A. Smith-Miles, 'A Meta-learning Approach to Automatic Kernel Selection for Support Vector Machines', *Neurocomputing*, **70**, 173–186, (2006).

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, 'Curriculum learning', in *Proceedings of the 26th International Conference on Machine Learning*, pp. 41–48. ACM, (2009).

[5] P. B. Brazdil, C. Soares, and J. Pinto Da Costa, 'Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results', *Machine Learning*, **50**(3), 251–277, (2003).

[6] G. Brown, J. L. Wyatt, and P. Tino, 'Managing diversity in regression ensembles.', *Journal of Machine Learning Research*, **6**, 1621–1650, (2005)

[7] M. S. Gashler, 'Waffles: A machine learning toolkit', *Journal of Machine Learning Research*, **MLOSS 12**, 2383–2387, (July 2011).

[8] T.A.F. Gomes and R.B.C. Prudêncio and C. Soares and A.L.D. Rossi and A. Cravalho, 'Combining Meta-learning and Search Techniques to Select Parameters for Support Vector Machines', *Neurocomputing*, **75**, 3–13, (2012).

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, 'The weka data mining software: an update', *SIGKDD Explorations Newsletter*, **11**(1), 10–18, (2009).

[10] T. K. Ho and M. Basu, 'Complexity measures of supervised classification problems', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**, 289–300, (March 2002).

[11] A. Hoekstra and R. P.W. Duin, 'On the nonlinearity of pattern classifiers', in *Proceedings of the 13th International Conference on Pattern Recognition*, pp. 271–275, (1996).

[12] L. I. Kuncheva and C. J. Whitaker, 'Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy.', *Machine Learning*, **51**(2), 181–207, (2003).

[13] J. Lee and C. Giraud-Carrier, 'A metric for unsupervised metalearning', *Intelligent Data Analysis*, **15**(6), 827–841, (2011).

[14] J. Lee and C. Giraud-Carrier, 'Automatic selection of classification learning algorithms for data mining practitioners', *Intelligent Data Analysis*, **17**(4), 665–678, (2013).

[15] A. Orriols-Puig, N. Macià, E. Bernadó-Mansilla, and T. K. Ho, 'Documentation for the data complexity library in c++', Technical Report 2009001, La Salle - Universitat Ramon Llull, (April 2009).

[16] A. H. Peterson and T. R. Martinez, 'Estimating the potential for combining learning models', in *Proceedings of the ICML Workshop on Meta-Learning*, pp. 68–75, (2005).

[17] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier, 'Meta-learning by landmarking various learning algorithms', in *Proceedings of the 17th International Conference on Machine Learning*, pp. 743–750, San Francisco, CA, USA, (2000). Morgan Kaufmann Publishers Inc.

[18] U. Rebbapragada and C. E. Brodley, 'Class noise mitigation through instance weighting', in *Proceedings of the 18th European Conference on Machine Learning*, pp. 708–715, (2007).

[19] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel, 'Automatic classifier selection for non-experts', *Pattern Analysis & Applications*, **17**(1), 83–96, (2014).

[20] M. Reif, 'A Comprehensive Dataset for Evaluating Approaches of Various Meta-learning Tasks', in *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pp. 273–276, (2012).

[21] M. R. Smith and T. Martinez, 'Improving classification accuracy by identifying and removing instances that should be misclassified', in *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 2690–2697, (2011).

[22] M. R. Smith and T. Martinez, 'A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems', *Computational Intelligence*, accepted, (2014).

[23] M. R. Smith, T. Martinez, and C. Giraud-Carrier, ‘An instance level analysis of data complexity’, *Machine Learning*, **95**(2), 225–256, (2014).
[24] J. Vanschoren, H. Blockeel, Bernhard Pfahringer, and Geoffrey Holmes, ‘Experiment databases - a new way to share, organize and learn from experiments’, *Machine Learning*, **87**(2), 127–158, (2012).

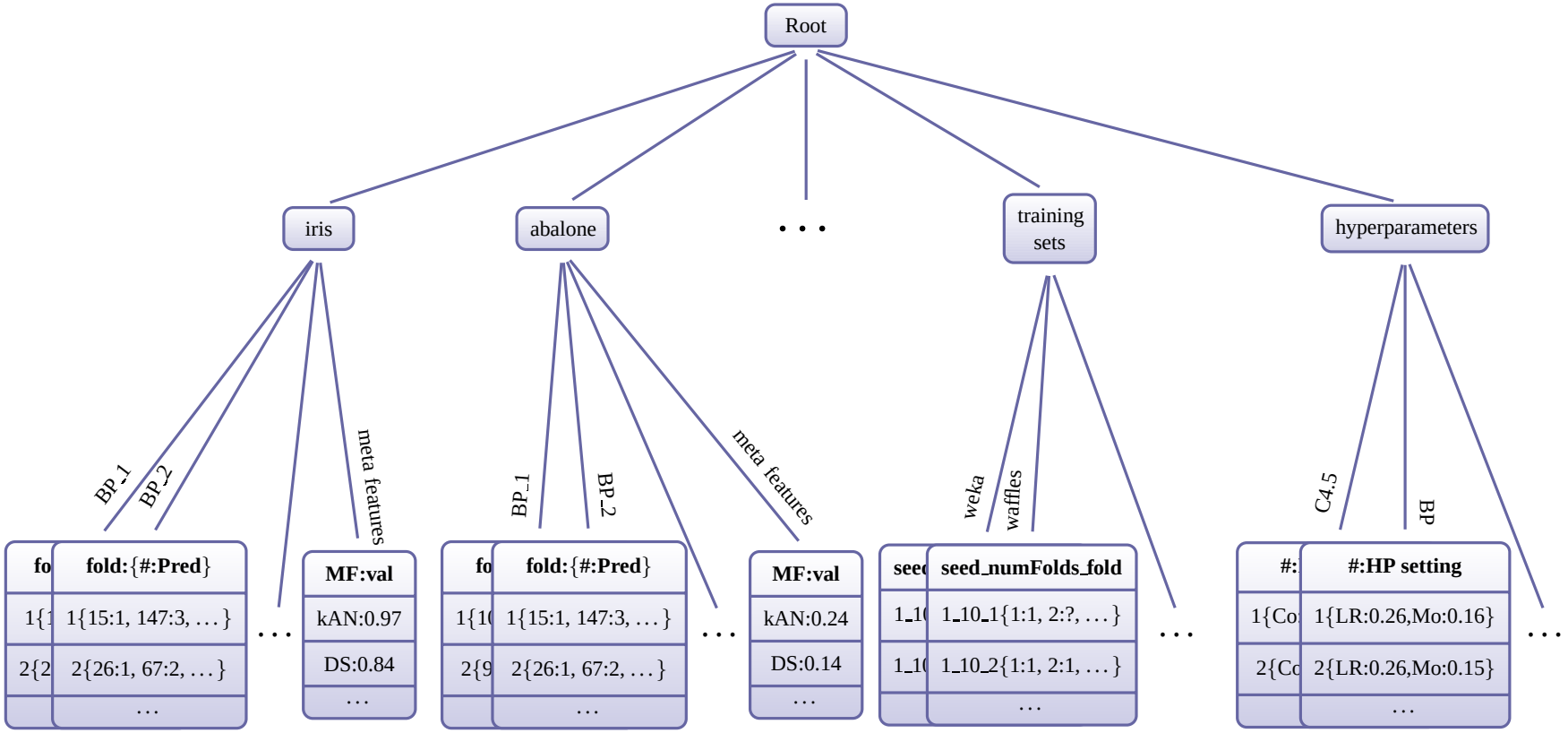

Figure 1. Hierarchical representation of how the results from machine learning experiments are stored in the NoSQL database for the MLRR. Each data set has a collection containing the predictions for each instance from a learning algorithm as well as its meta-features. A separate collection stores all of the information for the learning algorithms and which hyperparameters were used. Another collection stores the information for which instances were used for training.