Flexible Algorithm Selection Framework for Large Scale Metalearning

Eugene Santos Jr.[1], Alex Kilpatrick[1], Hien Nguyen[2], Qi Gu[1], Andy Grooms[1], Chris Poulin[1]

[1]Dartmouth College
Thayer School of Engineering.
8000 Cummings Hall
Hanover, NH 03755
{Eugene.Santos.Jr, Qi.Gu, Chris.Poulin}@Dartmouth.EDU
alex@tacticalinfosys.com , *agrooms@sbcglobal.net*

[2]University of Wisconsin-Whitewater
Dept. of Mathematical and Computer Sciences
800 W. Main Street.
Whitewater, WI 53190
nguyenh@uww.edu

Abstract— **We are working on the problem of developing a flexible, generic metalearning process that supports algorithm selection based on studying the algorithms' past performance behaviors. State of the art machine learning systems display limitations in that they require a great deal of human supervision to select an effective algorithm with corresponding options for a specific domain. Additionally, very little guidance is available for algorithm-parameter selection and the number of available choices is overwhelming. In this paper, we develop a flexible, large-scale experimental framework for a metacontroller that supports explorations through algorithm-parameter space and recommend algorithm for a given dataset. First, we aim to facilitate an easy to use process to create a search space for algorithm selection by automatically exploring some possible combinations of algorithms and key parameters. Secondly, our goal is to come up with an algorithm recommendation by looking at the past behaviors of related datasets. Our main contribution is the implemented framework itself which is based on the use of a wide variety of strategies to automatically generate a search space and recommend algorithms for a specific dataset. We evaluate our system with 40 major algorithms on 20 datasets from the UCI repository. Each dataset is represented by 25 data characteristics. We generate and run 7510 combinations of algorithm, parameters and datasets. Our experiments show that our framework offers a friendly way of setting up a machine learning experiment while providing accurate ranking of recommended algorithms based on past behaviors. Specifically, 88% of recommended algorithm rankings significantly correlated with the true rankings for a given dataset**.

Keywords: *metalearning, experimental study, algorithm selection.*

I. Introduction

Metalearning is the formal study of the best practices in machine learning [4] which enables the selection of optimal learning algorithms that best fit the search space of any given problem. Algorithm selection based on past performance is a very important task in metalearning, as shown in Rice's metalearning model [15]. The problem is defined as "For a given problem instance $x \in P$, with features $f(x) \in F$, find the selection mapping $S(f(x))$ into algorithm space A, such that the selected algorithm $\alpha \in A$ maximizes the performance mapping $y(\alpha(x)) \in Y$" [16]. This problem is often viewed as a search problem with the search space being the sets of machine learning algorithms performing with a set of parameters on a specific dataset. Users often encounter many algorithms, each containing many parameters, which, in turn may have many values. This makes the search space intractable for a complete search. Also there is very little guidance available for algorithm-parameter selection. One algorithm can be good for a specific problem but can perform poorly on another problem. Without domain knowledge, users can easily spend significant amounts of time in a suboptimal solution. This argument is also supported by the No Free Lunch theorem, as indicated in [16] which suggested that we should understand more about the datasets and the algorithms in order to choose an appropriate algorithm for a specific dataset and the task at hand.

In this paper, we empirically study the algorithm selection problem by developing a flexible metalearning framework to assist users in choosing appropriate learning algorithms for a specific dataset. The novelty of our approach is the design of the framework which allows for a wide variety of combinations between learning algorithms, their parameters and datasets. This framework enables any users to set up the experiments easily and automatically explores the algorithm-parameter search space to find the most appropriate algorithms. Our generic framework is built upon the Weka foundation [8], but is not tied to Weka. The framework is completely multi-threaded and parallelized to support execution on clusters. We use this framework to rank and recommend algorithms. In our experiment, we evaluated 40 classification algorithms categorized in six groups based on the way the classification is done (for example: using tree, rule or probability). Unlike previous algorithm recommender systems that focus on providing ranking based on uni-criterion evaluation, e.g. accuracy, we take both accuracy and running time into consideration while producing an algorithm ranking. Twenty datasets from the UCI repository [6] are used to assess the robustness of the system and in evaluating the recommended algorithm rankings. A series of recent studies have been conducted with regards to algorithm selection. For example, 112 datasets and 8 algorithms are used in the study of Ali and Smith [1]. However, most of these efforts only consider a limited number of classic algorithms, such as Support Vector Machine (SVM) and rule-based algorithms, whereas in our work, we exploit more classification algorithms from six different groups.

Each dataset in our testbed is described by 25 data characteristics extracted using a tool called DCT [11]. We use a leave-one-out strategy for assessing the recommended algorithm ranking. For each dataset in the testbed, we compute its similarity with all of the remaining datasets using the k-nearest neighbor (kNN) algorithm. Then we use the information of the algorithms performed on the nearest datasets to form a recommended algorithm list. The adjusted ratio of ratios (ARR) measure [5] is used to assess each algorithm performance. We compute the recommended ranking and the true ranking for each dataset, and then calculate the Spearman's correlation over these two ranking values. A true ranking of algorithms for a given dataset is the order in terms of the performance of each algorithm on the particular dataset, where the performance is evaluated via the overall ARR measurement. Moreover, we assessed the ranking based on the differences in terms of the end user's emphases. For example, machine learning practitioners may focus more on accuracy, whereas industry software developers may emphasize more on time. The results show that for all but one case, the recommended and true rankings correlated well with each other. More importantly, 88% of the recommended rankings are significantly correlated with the true ranking of algorithms for a given dataset.

This paper is organized as follows: We start out with a review on the study of algorithm selection as applied to classification problems in Section II. Section III details the algorithm and configuration of our framework. The experimental setup section then describes our objectives, and details of our testbed including datasets, algorithms, measures and the procedure. Section V discusses the findings from our results. Lastly, we will present our conclusion and future work.

II. Related Work

The aim of metalearning in general is to assist users in selecting the best performance algorithms in an appropriate model to solve a specific problem. The main research questions are: (i) which features of a dataset significantly affect the performance of a learning algorithm; and (ii) which algorithms are the most appropriate to solve a given problem. Substantive effort has been put into extracting meta-features, where the objective is to capture certain relationships between the measured data characteristics and the performance of the algorithms. Among those, StatLog project [12] is a comprehensive empirical study that provided 16 valuable meta-features that were widely used by metalearning researchers for many years. A number of studies extended the range of features by incorporating the structural characteristics of models [13] and applying simpler and faster learners such as landmarkers [14].

Regarding the selection of algorithms, one of the most comprehensive studies on meta-learning for algorithm selection is done by Smith-Miles [16]. Early meta-learning approaches such as StatLog, limit themselves to suggesting one or a subset of algorithms on a given problem. Brazdil et al. [5] extends StatLog by applying algorithm ranking rather than choosing the only best performance algorithm. The idea is to use a learning algorithm such as kNN to identify the similarity between datasets in the meta-feature space. Then the ranking of each algorithm was calculated based on its performance on the neighboring datasets. The adjusted ratio of ratios (ARR) which aggregates information concerning accuracy and time is used to compute the ranking for each algorithm. Another approach that supports multi-criteria ranking uses data envelopment analysis (DEA), which measures the efficiency by taking the ratio of total outputs to total inputs [2].

In 2004, Lai and Tsai. [10] conducted a study in which Naïve Bayes, term frequency inverted document frequency (TFIDF), kNN, and SVM are applied on the spam e-mail classification problem. It turned out that kNN did worst among these algorithms while the combination of Naïve Bayes and TFIDF gave better results than either alone. This study is of limited interest for metalearning, because of its narrow scope.

Recently, several general-purpose data mining packages, e.g. Weka [8], have been developed to assist the development of machine learning applications. They incorporate user-friendly graphical interface to facilitate the set-up, execution and subsequent analysis, but generally offer no real decision support to non-expert end-users. We build our generic framework upon the Weka foundation as it is an open source software consisting of an extensive collection of machine learning algorithms. From this, we further build and evaluate the flexible framework using ARR measures with characteristics extracted using DCT. Our main contributions are the flexibility of our implemented framework and the large number of classification algorithms. This framework allows us to add a combination of algorithm, parameter, dataset easily, automatically generates a search space and recommends algorithms for a specific dataset.

III. Framework

Our framework supports explorations through algorithm-parameter space. The framework is completely multi-threaded and parallelized to support execution on clusters. The key feature is an interface-based plug-in system which allows for easy integration of new algorithms. While the system follows most Weka conventions, it is a stand-alone system with no dependency on Weka code. The main components are shown in Fig. 1.

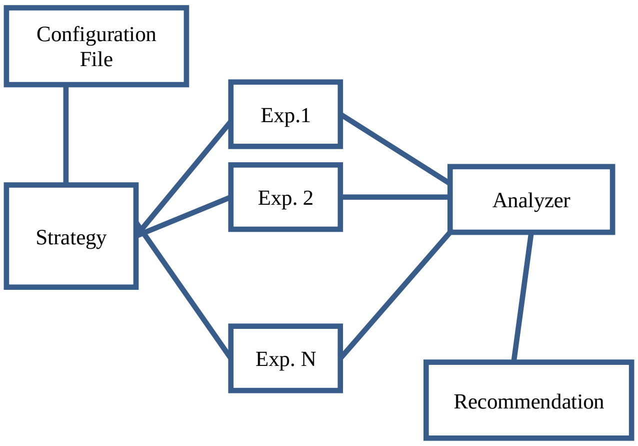

Figure 1: Architecture of algorithm selection framework.

This framework takes a configuration file as an input in which the description for each *task* in an experiment is included. A task is a learning problem described by a set of strategies, the input datasets, and the output result set. A *strategy* is defined as a tuple of a machine learning algorithm, a set of parameters and its corresponding values. If the set of parameters is ignored, the default parameters are used. Each parameter is described by type, tag, and values. Type can be numerical data type such as integer (denoted as INT), real values (denoted as REAL), boolean values (denoted as BOOL), or string. Tag represents the real name of the parameters. The values can be a single integer, real, boolean, string value or a range of values. We support numerical ranges of values by allowing the users to specify the lower bound value (denoted as *min*), upper bound value (denoted as *max*) and the step value (denoted as *step*). The system will automatically generate a loop through the values for a parameter. This feature distinguishes the framework from existing tools such as Experimenter in Weka where the users can only change a small set of parameters, as Weka does not support parameters with range values inside Experimenter.

Each parameter, in turn, can also be another machine learning algorithm which has its own sub-parameters. Fig. 2 shows an example of the configuration file which uses six small datasets from the UCI collection and BayesNet algorithm. A dataset is a collection of data organized in a certain format such as the ARFF format used by Weka. A dataset contains more than one instance in a specific domain such as labor, voting, census and so forth. An instance is defined as a row in a specific dataset which describes an observation of a known event in the past in that particular domain.

We use our own XML format to describe this framework with corresponding tags to present task and strategy. Even though our XML configuration file format is similar to XML format of Predictive Model Markup Language [7] (PMML), it gives us the flexibility and autonomy of adding any new algorithms into our framework without having to depend on PMML.

We provide a tool to convert a dataset from a non-Weka format to ARFF format. Our framework also allows an easy integration with a non-Weka classifier. We provide an interface with six methods to set training instances, test instances and run the classifier.

```
<?xml version="1.0" encoding="utf-8"?>
<tasks>
 <task name="My Task List" folds="10" out="results/flashmob_control1_new">
  <datasources>
           <data>anneal.arff</data>
           <data>arrythmia.arff</data>
           <data>audiology.arff</data>
           <data>autos.arff</data>
           <data>balance-scale.arff</data>
           <data>breast-cancer.arff</data>
   </datasources>
  <!-- Bayes algorithms -->
  <algorithm value="weka.classifiers.bayes.BayesNet">
   <option tag="D"  type="bool" value="true" />
   <sub tag="Q” class="weka.classifiers.bayes.net.search.local.SimulatedAnnealing">
    <suboption tag="A" value="1" min="1" max="10" step="1" />
    <suboption tag="U" min="4" max="20" step="2" />
   </sub>
   <sub tag="Q" value="weka.classifiers.bayes.net.search.local.K2">
    <suboption type="INT" tag="P" value="4" min="1" max="10" step="1" />
    <suboption tag="S" value="BAYES" />
    <suboption tag="S" value="MDL" />
    <suboption tag="S" value="AIC" />
   </sub>
   <sub tag="E" value="weka.classifiers.bayes.net.estimate.SimpleEstimator">
    <suboption type="REAL" tag="A" value="0.5" min="0.1" max="0.6" step="0.1"/>
   </sub>
  </algorithm>
 </task>
</tasks>
```

Figure 2: XML file contains description of learning algorithms.

The system is designed to allow for a wide variety of search strategies, via a callback interface. This architecture supports multiple strategies for algorithm-parameter space on multi-threaded and multi-processor environment. The algorithm-parameter space is extremely large in most cases, it is impossible to perform an exhaustive search. Therefore, on each learning run instance, a callback is made to an evaluator which then determines the next algorithm-parameter instance to search. This allows the system to dynamically change the strategy in real time as the exploration progresses. In the current implementation, a simplistic simulated annealing approach is used. Ideally, a configuration XML file provides the range of each parameter in order to limit the search space. The range is represented by the values which can represent the minimal and maximal values and the steps. However, the system will accommodate cases where this information is not provided. The system samples several instances in the span of available parameters for a particular algorithm and then uses a hill-climbing approach to refine the search, continuing past the best result in order to minimize the chance that a local maxima is used. It is important to realize that this is a very simplistic first-cut approach to a search strategy, and is not meant to be optimal. In future work, we plan to incorporate a more sophisticated machine learning algorithm to guide the search.

In the recommendation process, we use two approaches to suggest algorithms to the end users. If we have prior knowledge of the performance of algorithms on a given dataset, then algorithms are ranked by computing their overall ARR.

ARR [5] is defined as:

$$ARR^{d_i}{}_{a_p a_q} = \frac{\frac{SR^{d_i}_{a_p}}{SR^{d_i}_{a_q}}}{1+AccD*\log(\frac{T^{d_i}_{a_p}}{T^{d_i}_{a_q}})} \quad (1)$$

in which, $SR^{d_i}_{a_j} = 1 - ErrorRate(a_{j,}d_i)$ and $T^{d_i}_{a_j}$ is the running time of algorithm a_j on the dataset d_i. *AccD* is the tradeoff coefficient between time and accuracy of a specific algorithm. If the user emphasized more on accuracy, the value of *AccD* is small. If the user emphasized more on running time, the value of *AccD* is big. This measure aggregates information concerning accuracy and time for each algorithm against another algorithm in a specific dataset. The overall ARR for each algorithm on a given dataset is computed by taking the arithmetic mean of all ARRs of that algorithm against the remaining algorithms.

If we need to recommend algorithms for a new dataset which we do not have any prior knowledge, we will compute its similarity with the remaining datasets using the kNN algorithm applied on the data characteristics space. More detail is provided in Section IV. Although the system provides a recommendation for an algorithm-parameter set, extensive detailed output data from each run is recorded, supporting complex offline analysis or evaluation of output data though conventional machine learning algorithms.

IV. EXPERIMENTAL SETUP

The goals of this evaluation are twofold. First, we want to test the robustness of our implemented framework. Secondly, we assess the ranking of algorithms and compare the recommended ranking with the true ranking. We choose the classification problem to study in this experiment because this is a popular machine learning problem with well-established testbeds, procedure and measures.

A. Testbeds

We use forty algorithms which are categorized into six groups based on the ways a classifier can be constructed. These groups are Bayes (probability-based algorithms), functions (logic-based algorithms), lazy (distance-based algorithms), rule (rule-based algorithms), tree (tree-based algorithms) and a miscellaneous group. The list of algorithms is presented in Appendix 1. Twenty datasets from the UCI [6] repository have been used to evaluate the robustness and recommendation ranking of the framework.

Each dataset is represented by 25 data characteristics extracted using DCT tool [11]. These characteristics include general measurements such as number of features, number of instances; measurements of discriminant analysis applicable for numerical attributes such as *fract* and *cancor*, and information theoretical measurements for symbolic attributes such as class entropy, joint entropy. The list of datasets and some general characteristics are shown in the TABLE I. For more details, please see [11].

TABLE I. DATA CHARACTERISTICS OF TESTBED.

Collection	Features	Symbolic Features	Numerical Features	Instances	Classes
Anneal	39	32	6	898	6
Arrythmia	280	73	206	452	16
Audiology	70	69	0	226	24
Autos	26	10	15	205	7
Balance-scale	5	0	4	625	3
Breast-cancer	10	9	0	286	2
Car	7	6	0	1728	4
Cmc	10	7	2	1473	4
Credit-rating	16	9	6	690	2
Cylinder	40	21	18	540	2
Diabetes	9	0	8	768	2
Glass	10	0	9	214	7
Horse-Colic	23	15	7	368	2
Ionosphere	35	0	34	351	2
Iris	5	0	4	150	2
Labor	17	8	8	57	2
Segment	20	0	19	2310	7
Sonar	61	0	60	208	2
Splice	62	61	0	3190	3
Vote	17	16	0	435	2

We assess each strategy using twelve measures that are commonly used to assess machine learning algorithms. They are accuracy, running time, true positive, true negative, false positive, false negative, precision, recall, F-measure, specificity, information score, and root mean square error. Accuracy is defined as the ratio between the number of correctly classified instances over the number of all classified instances. Running time is defined as the difference between the ending and starting of the classification process. True positive is the number of correctly classified positive instances. True negative is the number of correctly classified negative instances. False negative is the number of incorrectly classified negative instances while false positive is the number of incorrectly classified positive instances. Precision is the ratio between the total number of correctly classified positive instances over the number of instances while recall is the ratio between the total number of correctly classified instances over the total number of positive instances. F-measure is computed as:

$$F - measure = \frac{2*Precision*Recall}{Precision+Recall} \quad (2)$$

Specificity is the percentage of negative instances that were predicted as negative. Information score is defined by [9] represents the prior information that is needed to correctly classify instances minus the residual information. Finally, root mean square error (RMS) is computed as the square root of the mean of the square of the differences of the predictions and the actual class of all instances.

B. Procedure

Our system automatically builds a set of strategies and performs classification on a specific dataset. We choose 10-fold Cross Validation (CV) to evaluate the algorithms, as it is a commonly used approach in machine learning for evaluating algorithm configuration. Basically, we partition the original dataset into 10 sub-sets, one of which is used for testing while the remaining nine sub-sets are used for training. The CV process is then run ten times and the twelve measures described above will be recorded and aggregated to produce a single measure for this strategy on this dataset. We refer to this step as the *learning algorithm step.* The results from this step contains the name of the dataset, name of algorithm, parameters and values of corresponding parameters, and the values of twelve measures described in subsection A for all combinations of the strategies and datasets. In the second step, named *algorithm selection step*, we need to come up with a list of recommended algorithms. Specifically, we define the *true ranking* for a dataset d_i as the ordering of algorithms based on their performance collected while classifying this dataset. This true ranking is computed as follows: first, the averages of the twelve values for each algorithm grouped by algorithm parameters are calculated. Then ${ARR^{d_i}}_{a_p a_q}$ is computed and the overall ARR for each algorithm is calculated as follows:

$${ARR^{d_i}}_{a_p} = \frac{1}{m}\sum_{a_q} {ARR^{d_i}}_{a_p a_q} \quad (3)$$

where *m* is the number of algorithms. The system ranked the algorithms by their ARRs in descending order. Since ${ARR^{d_i}}_{a_p a_q}$ depends on the value of *AccD* in addition to accuracy and running time. This rank is considered as the true ranking with a given value of *AccD* for the dataset d_i. The intuition behind this idea is that given two different users with two different preferences in the way they solve a learning problem, different lists of algorithms will be recommended.

If we do not know anything about the performance of any algorithms on a given dataset, kNN algorithm is used to find the nearest neighbor datasets. The set of algorithms performing well on these sets is used to generate the recommended list for the new dataset. We evaluate the ability to recommend algorithms by assessing the correlation between the true ranking obtained above and the recommended ranking. The recommended ranking on a specific dataset is computed by aggregating the ARR measurements on all the datasets in the nearest neighbor set. Basically, for each dataset in the testbed, we compute its Euclidean distance to each of the remaining datasets. Each dataset is described by 25 data characteristics presented earlier in subsection A. Given two datasets $a_p=\{p_1,p_2,\ldots,p_{25}\}$ and $a_q=\{q_1, q_2,\ldots,q_{25}\}$, the Euclidean distance between a_p and a_q is defined as follows:

$$distance(a_p, a_q) = \sqrt{\sum_{i=1}^{25}(p_i - q_i)^2} \quad (4)$$

Note that these values p_i and q_i have been normalized. Then *k* datasets with smallest distances are chosen to be the nearest neighbors. In this implementation, we choose k=3. After the nearest neighbor set is formed, we compute the ranking by aggregating the ARR through all of the datasets in the nearest neighbor sets and taking averages of all available algorithms. The approach is discussed in details in [5].

$$ARR_{a_p} = \frac{1}{m}\sum_{aq} \sqrt[k]{\prod_{d_i} ARR^{d_i}\, a_p a_q} \quad (5)$$

in which *m* is the number of algorithms and *k* is the number of datasets in the nearest neighbor set.

We compute the true ranking and recommended ranking for three values of *AccD*. Recalling that AccD is the value that represents the emphasis the user has on either time or accuracy. We choose three values for AccD (0.1%, 1% and 10%) as they are used in other experiments with ARR [5]. With *AccD* being 0.1%, it reflects a profile of a machine learning practitioner who emphasizes more on accuracy while with *AccD* being 10%, it reflects a profile of an industry software developer who emphasizes more on running time. After the true ranking and recommended ranking are computed, we use Spearman's correlation to find if the rankings are correlated with each other. Since we are more interested in ranking order (ordinal scale), Spearman correlation is an appropriate choice for analysis. All data analysis is performed using SPSS version 9.0 [17].

V. RESULTS AND DISCUSSION

The goals of our evaluation are addressed through this experiment. First, the robustness and ease of use of our framework has been tested with 7510 runs. Each run is the execution of a strategy on a dataset on one node of our cluster: Dell Poweredge 2950 with 2 quad core Intel Xeon E5420 processors @ 2.50GHz, 16GB Ram, 80GB 7.2k rpm SATA drive. The input of each run is a tuple (algorithm, parameters, dataset). The output of each run is the value of the above 12 measures. The outputs of this step then serve as inputs to the algorithm selection step, after which, an algorithm ranking is produced. This ranking is called true ranking for a given dataset for a given user profile. TABLE II shows the real ranking of the top 5 of algorithms for each dataset in the testbed.

TABLE II. TOP 5 ALGORITHMS FOR ALL DATASETS.

Dataset	**Top 5 algorithms for accuracy emphasized users (AccD=0.1%)**	**Top 5 algorithms for neutral users (AccD=1%)**	**Top 5 algorithms for running time emphasized users (AccD=10%)**
Anneal	LMT,RandomTree NNge,HyperPipes, JRIP	NNge, RandomTree, SMO-PolyKernel, RBFNetwork, JRIP	ZeroR, NNge, RandomTree, SMO-PolyKernel, OneR
Arrythmia	ADTree,VotedPerceptron, SPegasos, AODE, LBR	VotedPerceptron, ADTree, Spegasos, AODE, LBR	ZeroR, DecisionStump, SimpleLogistic, OneR, RandomTree
Audiology	IB1,SMO-PolyKernel,LMT, HyperPipes, NNge	ZeroR, IB1, OneR, DecisionStump, ConjunctiveRule	OneR, IB1, DecisionStump, ConjunctiveRule, RandomTree
Autos	LMT, RandomTree, NNge, HyperPie, JRIP	NNge, RandomTree, SMO_PolyKernel , RBFNetwork, JRIP	ZeroR, NNge, RandomTree, SMO-PolyKernel, OneR
Balance-scale	LMT, FT, RandomForest, NNge, RandomTree	FT, NNge, LMT, SMO-PolyKernel,IB1	RBFNetwork, NBTree, DecisionStump, LADTree, DecisionTable-BestFirst
Breast-cancer	IB1, RandomForest, RandomTree, NNge, KStar	IB1, RandomForest, NNge, RandomTree, JRIP	BFTree,, DecisionStump, ZeroR, NNge, RandomForest
Car	NNge, BFTree, SimpleCart, LMT, SMO-PolyKernel	NNge, IB1, RandomTree, RBFNetwork, BFTree	IB1, OneR, NNge, DecisionStump, ZeroR
Cmc	RBFNetwork, Logistic, RandomForest, RandomTree, NNge	RBFNetwork, Logistic, RandomTree, RandomForest, IB1	ZeroR, RBFNetwork, Logistic, HyperPiper, RandomTree
Credit-rating	RandomForest, NNge, IB1, RandomTree, JRIP	NNge, RandomForest, IB1, BFTree, RandomTree	LADTree, BFTree, OneR, SMO-PolyKernel, DecisionStump
Cylinder	Logistic, KStar, HyperPipes, IB1, NNge	HyperPipes, Logistic, NNge, KStar, RandomTree	HyperPipes, VFI, NaiveBayesUpdateable, ZeroR, RandomTree
Diabetes	RandomForest, NNge, RandomTree, IB1, BFTree	RandomForest, NNge, RandomTree, BFTree, REPTree	BFTree, ADTree, REPTree, NNge, RandomForest
Glass	IB1, RandomForest, Logistic, LMT, RandomTree	NNge, IB1, SMO_PolyKernel , RandomTree, RBFNetwork	IB1, NNge, DecisionStump, ZeroR, OneR
Horse-Colic	Winnow, LBR, AODE, RandomForest, BFTree	Winnow, LBR, AODE, SimpleCart, RandomForest	VotePerceptron, BFTree, NNge, RandomTree, JRIP
Ionosphere	RandomForest, NNge, RandomTree, JRIP, NBTree	NNge, BFTree, RandomTree, JRIP, RandomForest	NNge, OneR, BFTree, JRIP, RandomTree
Iris	NNge, RandomTree, IB1, LADTree, RandomForest	NNge, BFTree, RandomTree, JRIP, REPTree	NNge, OneR, BFTree, JRIP, RandomTree
Labor	NaiveBayes, JRIP, Logistic, LMT, RBFNetwork	NaiveBayes, JRIP, LMT, RBFNetwork, NNge	BFTree, JRIP, NaiveBayes, NNge, OneR
Segment	HyperPipes, SimpleCart, IB1, BFTree, NNge	BFTree, SimpleCart, HyperPipes, NNge, IB1	ZeroR, BFTree, SimpleCart, HyperPipes, OneR
Sonar	IB1, RandomForest, NBTree, JRIP, RandomTree	IB1, RandomForest, LADTree, ADTree, RandomTree	LADTree, ADTree, FT, RandomTree, ZeroR
Splice	AODE, FT, SMO-PolyKernel, NaiveBayesSimple, JRIP	NaiveBayesSimple, VFI, AODE, SMO-Kernel, RandomForest	NaiveBayesSimple, ZeroR, OneR, HyperPipes, DecisionStump
Vote	RandomForest, NNge, ADTree, RandomTree, IB1	RandomForest, RandomTree, IB1, NNge, ADTree	OneR, DecisionStump, RandomTree, IB1, RandomForest

In TABLE III, we show an example of the accuracy and running time (in milliseconds) for the top 5 algorithms performed on the *anneal* dataset. With AccD value being 10%, the algorithms with shorter running time are preferred. Therefore, we can see the algorithms with faster average running time such as ZeroR, NNge, Random Tree. With value of AccD being 0.1%, the algorithms with higher accuracy are preferred. Thus, we saw algorithms with faster average running time such as LMT in this list.

TABLE III. AN EXAMPLE OF RUNNING TIME AND ACCURACY TRADEOFF FOR *ANNEAL* DATASET.

AccD=10% (emphasize on running time)			AccD=0.1% (emphasize on accuracy)		
Algorithm	Accuracy	Time	Algorithm	Accuracy	Time
ZeroR	0.327	2	LMT	0.902	1131
NNge	0.887	6	RandomTree	0.884	13
RandomTree	0.884	13	Nnge	0.887	6
SMO-PolyKernel	0.833	13	HyperPipes	0.744	91
OneR	0.553	9	JRIP	0.890	77

We assess the second goal of our evaluation by computing the correlation between true rankings and the recommended rankings. The recommended ranking is created by aggregating the ranking of all the algorithms performed with all the datasets in the nearest neighbor sets. The nearest neighbor set for a specific dataset is computed by finding *k* datasets (k=3 in this implementation) that have the shortest Euclidean distance to that given dataset. TABLE IV shows the datasets and their nearest neighbor sets. NN1, NN2, NN3 are the name of the datasets in the nearest neighbor for a given dataset and distance 1, distance 2, and distance 3 are the corresponding Euclidean distance from NN1, NN2, and NN3 to the given dataset.

We generated the recommended algorithm list for all datasets and computed the Spearman's correlation between the true ranking and the recommended ranking of algorithms for a specific dataset. The results are shown in TABLE V.

TABLE IV. DATASETS AND THEIR NEAREST NEIGHBORS

Dataset	NN1	Distance	NN2	Distance 2	NN3	Distance 3
anneal	vote	0.420	labor	0.431	balance-scale	0.432
arrhythmia	audiology	0.473	sonar	0.526	cylinder	0.537
audiology	autos	0.361	glass	0.456	arrhythmia	0.473
autos	vote	0.261	car	0.264	breast-cancer	0.342
breast-cancer	diabetes	0.073	ionosphere	0.083	sonar	0.153
car	diabetes	0.119	credit-rating	0.151	cylinder	0.162
cmc	Ionosphere	0.170	breast-cancer	0.187	diabetes	0.199
credit-rating	ionosphere	0.081	sonar	0.112	vote	0.121
cylinder	car	0.162	vote	0.171	sonar	0.181
diabetes	breast-cancer	0.073	ionosphere	0.083	vote	0.114
glass	iris	0.188	balance-scale	0.228	segment	0.229
horse-colic	vote	0.090	ionosphere	0.139	labor	0.170
ionosphere	labor	0.068	credit-rating	0.081	breast-cancer	0.083
iris	ionosphere	0.171	glass	0.188	vote	0.205
labor	Ionosphere	0.068	vote	0.088	disbetes	0.127
segment	glass	0.229	cmc	0.263	iris	0.302
sonar	credit-rating	0.112	breast-cancer	0.153	vote	0.161
splice	car	0.277	cmc	0.313	cylinder	0.316
vote	labor	0.088	horse-colic	0.090	diabetes	0.114

TABLE V. SPEARMAN'S CORRELATION BETWEEN REAL RANKING AND RECOMMENDED RANKING OF ALGORITHMS FOR ALL 20 DATASETS.

	AccD=10%		AccD=0.1%		AccD=1%	
Dataset	Spearman's correlation	Sig (1-tail)	Spearman's correlation	Sig (1-tail)	Spearman's correlation	Sig (1-tail)
Anneal	0.457	0.012	0.525	0.004	0.575	0.001
Arrhythmia	0.208	0.14	0.144	0.224	-0.166	0.19
Audiology	0.302	0.071	0.314	0.059	0.067	0.373
Autos	0.446	0.011	0.4	0.021	0.337	0.046
Balance-scale	0.148	0.231	0.57	0.001	0.504	0.004
Breast-cancer	0.383	0.018	0.68	<0.001	0.68	<0.001
Car	0.541	0.001	0.8	<0.001	0.545	0.001
CMC	0.333	0.045	0.57	0.001	0.5	0.004
Colic	0.316	0.044	0.745	<0.001	0.618	<0.001
Credit-rating	0.54	0.001	0.694	<0.001	0.656	<0.001
Cylinder	0.573	<0.001	0.597	0.001	0.501	0.002
Diabetes	0.372	0.028	0.784	<0.001	0.805	<0.001
Glass	0.703	<0.001	0.742	<0.001	0.721	<0.001
Ionosphere	0.338	0.039	0.829	<0.001	0.639	<0.001
Iris	0.659	<0.001	0.73	<0.001	0.833	<0.001
Labor	0.333	0.045	0.477	0.006	0.463	0.008
Segment	0.742	<0.001	0.743	<0.001	0.761	<0.001
Sonar	0.516	0.002	0.794	<0.001	0.801	<0.001
Splice	0.655	<0.001	0.481	0.007	0.375	0.032
Vote	0.669	<0.001	0.831	<0.001	0.693	<0.001

As we can see, among 60 pairs of algorithm rankings, there is only one pair (dataset: *arrhythmia*, AccD=1%) that yields negative correlation. Six more pairs have weak correlations (Spearman's correlation < 0.3, Sig(1-tail) > 0.05). The remaining fifty three pairs (88%), as highlighted in TABLE V, have correlations bigger than 0.3 (Sig (1-tail) <0.05). More importantly, thirty nine pairs (65%) have strong correlation (Spearman's correlation>= 0.5, Sig (1-tail) < 0.05). Notice that the distances of the three datasets in the nearest neighbors of *arrhythmia* dataset are bigger than other distances for all three values of AccD. That means using the algorithm rankings for *audiology*, *sonar* and *cylinder* to predict the ranking for *arrhythmia* may not be precise. Additionally, the distance between two dataset depends on the set of data characteristics. Therefore, the more related the data characteristic set is to the algorithm performance, the more precise and more significant the distance is.

In summary, our framework has allowed us to set up experiments very easily, especially when dealing with parameters having ranges of values. More importantly, it produces the algorithms rankings related to the true ranking for a given dataset.

VI. CONCLUSION

We have reported our effort on the development of a flexible, generic framework that supports algorithm selection based on studying the algorithms' past performance behaviors on relevant datasets. In this work, we considered all the features of the datasets as well as all data characteristics of the datasets for determining the relevant datasets. Our contributions are two folds. First, the framework is very flexible for users to add any news algorithms and parameters. Additionally, it supports parameters with range values. Secondly, we use the typical evaluation procedure in metalearning to assess the algorithm rankings. Our results show that 88% of the recommended rankings correlate with the true rankings. As there is very little guidance available for algorithm-parameter selection, this framework can be used to automatically generate the search space and suggest the algorithm depending the user's preferences.

This problem is particularly interesting and challenging in both the research and validation phases. The development and evaluation of this framework is the beginning of our quest to explore the combination between algorithm selection and feature selection in metalearning. There are many ways we can extend and improve this framework. In this work, we currently use 25 data characteristics extracted by DCT tool to represent a dataset. The problem is that except for the general set of characters, some datasets with only numerical features may not have the same data characteristics with the datasets having only symbolic features. We would like to explore other types of data characteristics such as DeCT [13] which explore the tree structures of datasets. Secondly, we currently use the whole set of features of any dataset in our testbed in our learning algorithm. The machine learning community has a long history of work on feature selection that can be used to improve the process. Next, one direction closely related to metalearning is deep learning [3] which is a prominent form of hierarchical machine learning. Deep learning utilizes many deep layers of abstract representation, inspired by human visual processing capabilities. Deep learning may help us discover the semantic meaning behind why certain algorithms perform best on some testbeds and poorly in others. These two themes (deep learning and metalearning) ultimately come down to the *intentional* use of structured signal similarity/intersection (A ∩ B) and the unstructured noise dis-similarity/symmetric difference. As such, we would argue for a 'Relative Network' of objects and their relationships, structured entirely along the above. We would like to leverage our selection algorithm work to understand deeply why certain algorithms work on certain testbeds but poorly on others.

ACKNOWLEDGEMENT

This research is a part of the Dartmouth Metalearning Working Group project, award N10PC20221, funded by IARPA. Thank you Christopher Witt for useful comments while editing this paper.

REFERENCES

[1] S. Ali, and K. Smith. "On learning algorithm selection for classification". Applied Soft Computing, 6(2), pp. 119–138. 2006.

[2] A. Bazleh, P. Gholami and F. Soleymani. "A New Approach Using Data Envelopment Analysis for Ranking Classification Algorithms". Journal of Mathematics and Statistics, 7 (4), pp. 282-288, 2011.

[3] Y. Bengio. "Learning Deep Architectures for AI". Foundations and Trends in Machine Learning. 2(1), pp. 1-127. 2009.

[4] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta. "Metalearning: Application to Data Mining". Springer. 2010.

[5] P. B. Brazdil, C. Soares, J. P. DA COSTA. "Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results". Machine Learning, 50, pp. 251–277, 2003.

[6] A. Frank & A. Asuncion. "UCI Machine Learning Repository". Irvine, CA: University of California, School of Information and Computer Science. 2010.

[7] A.Guazzelli, M. Zeller, W. Chen, and G. Williams. "PMML: An Open Standard for Sharing Models". The R Journal. 1/1, 2009

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. "The WEKA data mining software: an update". SIGKDD Explore. Newsletter. 11, 1, pp. 10-18. 2009.

[9] I. Kononenko and I. Bratko. "Information-based evaluation criterion for classifier's performance". Machine Learning, 6(1), pp. 67–80. 1991.

[10] C-C, Lai, and M-C Tsai. "An Empirical Performance Comparison of Machine Learning Methods for Spam E-mail Categorization". In Proceedings of the Hy-brid Intelligent Systems, HIS '04, pp. 44 - 48. 2004

[11] G. Lindner and R. Studer. "AST: Support for Algorithm Selection with a CBR Approach". In Proceedings of PKDD '99, Jan M. Zytkow and Jan Rauch (Eds.). Springer-Verlag, pp. 418-423.1999.

[12] D. Michie, D. J. Spiegelhalter, and C. Taylor Eds. "Machine Learning, Neural and Statistical Classification". Ellis Horwood, New York. 1994

[13] Y. Peng, P. A. Flach, C. Soares and P. Brazdil. "Improved Dataset Characterisation for Meta-learning". Lecture Notes in Computer Science, 2534/2002, pp. 193-208. 2002.

[14] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. "Meta-Learning by landmarking various learning algorithms". In Proceedings of the 17th International Conference on Machine Learning, pp. 743-750. 2000.

[15] J. R. Rice, "The algorithm selection problem," ser. Advances in Computers, M. Rubinoff and M. C. Yovits, Eds. Elsevier, 15, pp. 65 – 118. 1976.

[16] K. A. Smith-Miles. "Cross-disciplinary perspectives on meta-learning for algorithm selection". ACM Computing. Survey. 41(1), Article 6, 2009.

[17] SPSS Inc. SPSS Base 10.0 for Windows User's Guide. SPSS Inc., Chicago IL. 1999.

Appendix 1

List of algorithms in the experiments

Bayes algorithms	**Lazy algorithms**
AODE	IB1
AODEsr	IBk
NaiveBayes	KStar
NaiveBayesSimple	LBR
NaiveBayesUpdateable	LWL
BayesNet	
Function algorithms	Rule algorithms
Logistic	ConfunctiveRule
MultilayerPerceptron	DecisionTable
RBFNetwork	DTNB
SimpleLogistic	JRIP
SMO	NNge
SPegasos	OneR
VotedPerceptron	ZeroR
Winnow	
Tree algorithms	
ADtree	LMT
BFTree	NBTree
DecisionStump	RandomForest
FT	RandomTree
J48	REPTree
LADTree	SimpleCart
Misc algorithms	
HyperPipes	VFI