Journal of International Technology and Information Management

Volume 23 | Issue 1 Article 1

2014

A Modified Stacking Ensemble Machine Learning Algorithm Using Genetic Algorithms

Riyaz Sikora
The University of Texas at Arlington

O'la Hmoud Al-laymoun
The University of Texas at Arlington

A Modified Stacking Ensemble Machine Learning Algorithm Using Genetic Algorithms

Riyaz Sikora
O'la Hmoud Al-laymoun
Department of Information Systems
The University of Texas at Arlington
USA

ABSTRACT

With the massive increase in the data being collected as a result of ubiquitous information gathering devices, and the increased need for doing data mining and analyses, there is a need for scaling up and improving the performance of traditional data mining and learning algorithms. Two related fields of distributed data mining and ensemble learning aim to address this scaling issue. Distributed data mining looks at how data that is distributed can be effectively mined without having to collect the data at one central location. Ensemble learning techniques aim to create a meta-classifier by combining several classifiers created on the same data and improve their performance. In this paper we use concepts from both of these fields to create a modified and improved version of the standard stacking ensemble learning technique by using a genetic algorithm (GA) for creating the meta-classifier. We use concepts from distributed data mining to study different ways of distributing the data and use the concept of stacking ensemble learning to use different learning algorithms on each sub-set and create a meta-classifier using a genetic algorithm. We test the GA-based stacking algorithm on ten data sets from the UCI Data Repository and show the improvement in performance over the individual learning algorithms as well as over the standard stacking algorithm.

INTRODUCTION

According to some estimates we create 2.5 quintillion bytes of data every day, with 90% of the data in the world today being created in the last two years alone (IBM (2012)). This massive increase in the data being collected is a result of ubiquitous information gathering devices, such as sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. With the increased need for doing data mining and analyses on this big data, there is a need for scaling up and improving the performance of traditional data mining and learning algorithms. Two related fields of distributed data mining and ensemble learning aim to address this scaling issue. Distributed data mining looks at how data that is distributed can be effectively mined without having to collect the data at one central location (Zeng et al., 2012). Ensemble learning techniques aim to create a meta-classifier by combining several classifiers, typically by voting, created on the same data and improve their performance (Dzeroski & Zenko, 2004; Opitz & Maclin, 1999). Ensembles are usually used to overcome three types of problems associated with base learning algorithms: the statistical problem; the computational problem; and the representational problem (Dietterich, 2002). When the sample size of a data set is too small in comparison with the possible space of hypotheses, a learning algorithm might choose to output a hypothesis from a set of hypotheses having the same accuracy on the training data. The statistical problem arises in such cases if the chosen hypothesis cannot predict new data. The computational problem occurs when a learning algorithm gets stuck in a wrong local minimum instead of finding the best hypothesis within the hypotheses space. Finally, the representational problem happens when no hypothesis within the hypotheses space is a good approximation to the true function *f*. In general, ensembles have been found to be more accurate than any of their single component classifiers (Opitz & Maclin, 1999; Pal, 2007)).

The extant literature on machine learning proposes many approaches regarding designing ensembles. One approach is to create an ensemble by manipulating the training data, the input features, or the output labels of the training data, or by injecting randomness into the learning algorithm (Dietterich, 2002). For example, Bagging learning ensembles, or bootstrap aggregating, introduced by Breiman (1996), generates multiple training datasets with the same sample size as the original dataset using random sampling with replacement. A learning algorithm is then applied on each of the bootstrap samples and the resulting classifiers are aggregated using a plurality vote when predicting a class and using averaging of the prediction of the different classifiers when predicting a numeric value. While Bagging can significantly improve the performance of unstable learning algorithms such as neural networks, it can be ineffective or even slightly deteriorate the performance of the stable ones such as k- nearest neighbor methods (Breiman, 1996).

An alternative approach is to create a generalized additive model which chooses the weighted sum of the component models that best fit the training data. For example, Boosting methods can be used to improve the accuracy of any “weak” learning algorithm by assigning higher weights for the misclassified instances. The same algorithm is then reapplied several times and weighted voting is used to combine the predictions of the resulting series of classifiers (Pal, 2007). Examples of Boosting methods include AdaBoost, AdaBoost.M1 and AdaBoost.M2 which were proposed by Freund & Schapire (1996). In a study conducted by Dietterich (2000) comparing the performance of the three ensemble methods Bagging, Randomizing and Boosting using C4.5 on 33 datasets with little or no noise, AdaBoost produced the best results. When classification noise was added to the data sets, Bagging provided superior performance to AdaBoost and Randomized C4.5 through increasing the diversity of the generated classifiers. Another approach is to apply different learning algorithms to a single dataset. Then the predictions of the different classifiers are combined and used by a meta-level-classifier to generate a final hypothesis. This technique is called “stacking” (Dzeroski & Zenko, 2004).

This article uses concepts from ensemble learning and distributed data mining to create a modified and improved version of the stacking learning technique by using a genetic algorithm (GA) for creating the meta-classifier. We use WEKA (http://www.cs.waikato.ac.nz/ml/weka/), the suite of machine learning and data mining algorithms written in Java for all our experiments. We use concepts from distributed data mining to study different ways of distributing the data and use the concept of stacking ensemble learning to use different learning algorithms on each sub-set and create a meta-classifier using a genetic algorithm. We test the GA-based stacking algorithm on ten data sets from the UCI Data Repository (http://archive.ics.uci.edu/ml/) and

show the improvement in performance over the individual learning algorithms as well as over the standard stacking algorithm.
The rest of the paper is organized as follows: the stacking ensemble learning approach; the modified stacking algorithm using genetic algorithm; the data sampling and decomposition techniques used; the results and discussion; and the conclusion.

STACKING ENSEMBLE LEARNING

In the standard stacking algorithm shown in figure 1, *n* different subsets of the training data set are created by using stratified sampling with replacement in which the relative proportion of the different classes is maintained in all the subsets. Each subset of the training set used to determine the performance of the classifiers on the training set. A meta classifier in the form of relative weight for each classifier is created by assigning a weight to a classifier that is proportional to its performance.

When evaluating an instance from the test set, every classification algorithm in WEKA gives a class distribution vector for that instance that gives the probability of that particular instance belonging to a given class. We can represent the class distribution vector over *c* classes for the *j*th classifier by a 1 x *c* vector as follows:

$$\Delta_j = [\delta_{1j} \;\; \delta_{2j} \;\; \dots \;\; \delta_{cj}] \quad 1 \le j \le n \tag{1}$$

where,

$$0 \le \delta_{ij} \le 1 \;\; \forall 1 \le i \le c$$
$$\sum_i \delta_{ij} = 1$$

The class distribution vectors for the *n* classifiers can then be represented by an *n* x *c* matrix as follows:

$$\mathbf{\Delta} = [\Delta_1 \;\; \Delta_2 \;\; \dots \;\; \Delta_n]^{\mathrm{T}} \tag{2}$$

The meta-classifier creates a weight distribution vector that gives relative weight to different classifiers. The weight distribution vector over *n* classifiers is represented as follows:

$$\Theta = [\theta_1 \;\; \theta_2 \;\; \dots \;\; \theta_n] \tag{3}$$

where,

$$0 \le \theta_j \le 1$$
$$\sum_j \theta_j = 1$$

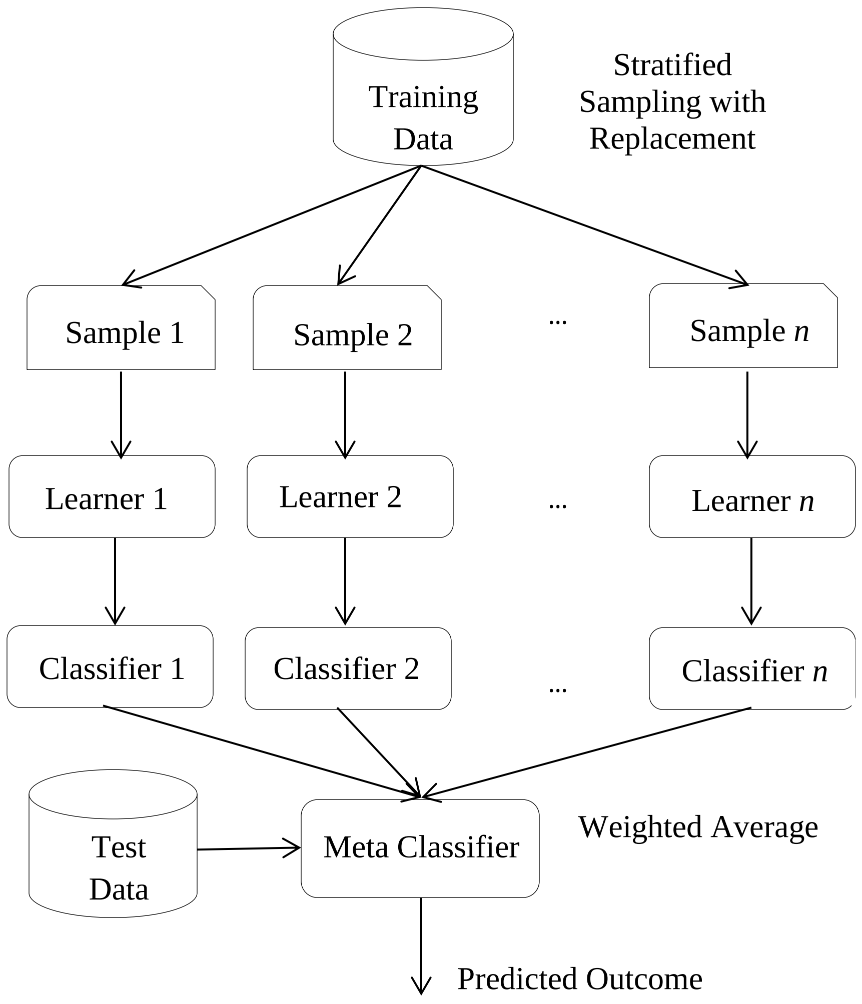
Given the class distribution matrix and the weight distribution vector, the meta-classifier evaluates each instance of the test set by using the following 1 x *c* class distribution vector:

$$\Delta' = \Theta . \mathbf{\Delta} = [\delta_1' \;\; \delta_2' \dots \;\; \delta_c'] \tag{4}$$

where,

$$\delta_i' = \sum_j \theta_i \delta_{ij}$$

Figure 1. Standard stacking ensemble learning.

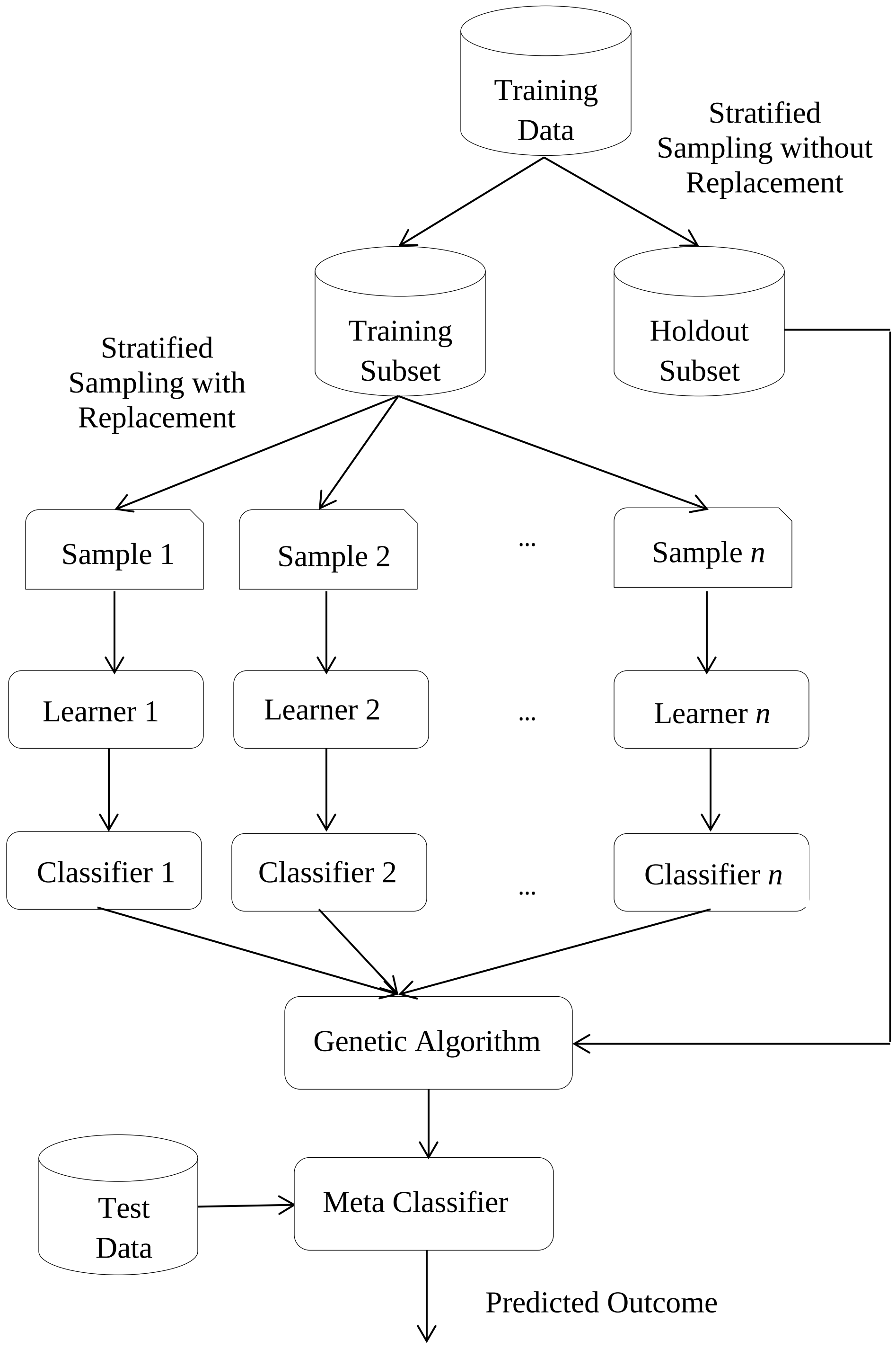


As mentioned above, in the standard stacking algorithm the meta-classifier weight distribution vector Θ is created by assigning a weight to a classifier that is proportional to its performance. In the next section we discuss using a genetic algorithm to learn the weight distribution vector.

GENETIC ALGORITHM BASED STACKING ENSEMBLE LEARNING

Genetic Algorithms (GAs) (Goldberg, 1989) combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm. GAs have been used in machine learning and data mining applications (Aci et al., 2010; Freitas, 2002; Agustin-Blas et al., 2012; Sikora &Piramuthu, 2005). GAs have also been used in optimizing other learning techniques, such as neural networks (Sexton *et al.*, 2003).

Figure 2. Stacking ensemble learning using genetic algorithm as meta learner.



The stacking ensemble learning using genetic algorithm as the meta-learner is shown in figure 2. The training data set is split into a training subset and a holdout subset. The training subset is further split into n subsets using stratified sampling with replacement, which are used by different learning algorithms to create n classifiers. The genetic algorithm is then used to learn a weight distribution vector that creates the meta classifier for predicting the test set instances.

In our case, the GA implements a weight distribution vector Θ as an individual member of the population. Each population member is therefore a vector of weights for each classifier, that all add up to 1.0. Based on some initial set of experiment runs we chose the following operators and parameter values for the GA. We used tournament selection of size 2 as the selection operator, a standard one-point crossover operator, and a mutation operator where one value from the vector of weights for an individual is randomly changed by a small amount. When an operator creates an invalid vector, i.e., whose weights do not add up to 1.0, we simply normalize the vector by dividing each weight value by the sum of all weights. We used a population size of 30 and the probabilities of crossover and mutation as 0.7 and 0.1 respectively. Note that the aim of our study was not to find the optimal parameter settings for the GA. In most cases the optimum settings would vary with data sets. Instead, our goal is to show the efficacy of this modified algorithm in general.

The GA begins by creating a random population of weight distribution vectors. The evaluation of each population member is done by evaluating the corresponding meta-classifier created by using its weight distribution vector on the holdout subset. The fitness of each member is then calculated to be the prediction accuracy of that meta-classifier on the holdout subset. Using the fitness of each population member, the GA then performs the tournament selection to select members for the next generation. It then applies the crossover and mutation operators to create a new generation of weight distribution vectors. The above process is repeated for 3000 generations, and the best weight distribution vector from its final population is selected to create the meta-classifier.

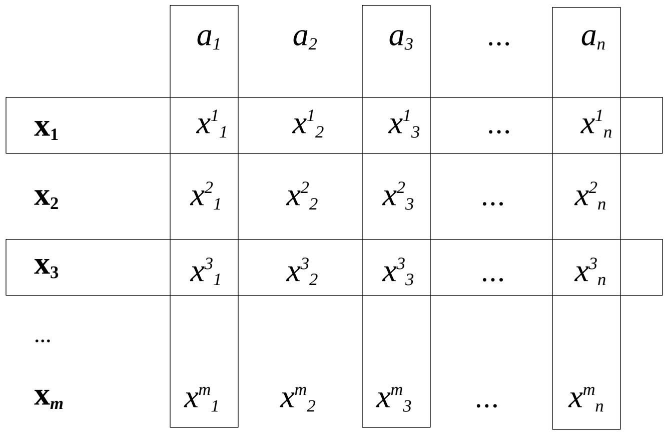
In the next section we give details about the data sampling and data decomposition techniques that were applied.

DATA SAMPLING AND DECOMPOSITION

The data sampling and decomposition shown in figures 1 and 2 can be done either along instances or along attributes as depicted in figure 3. In the instance-based decomposition, each sample receives only a subset of the total instances from the original data set. In the attribute-based decomposition, each sample receives a subset of the attributes from the original data set. We use two parameters to control the type and amount of decomposition. For instance-based decomposition we use the parameter $0 < pEx \leq 1$ that gives the proportion of the examples/instances that are selected for each subset. For attribute-based decomposition we use the parameter $0 < pAtt \leq 1$ that gives the proportion of the attributes that are selected for each subset.

These two data decomposition techniques also have practical implications for distributed data mining. In many scenarios data is naturally distributed and it is infeasible or impractical or insecure to collect all the data at one site for data mining. In such cases, it is important to do local data mining at the individual sites and then integrate the results. In some cases, the number of attributes might be too large for a standard learning algorithm to handle. By showing the efficacy of the stacking method presented in this paper, we also provide an efficient mechanism of doing distributed data mining in such instances.

Figure 3. Instance-based and attribute-based decomposition.


a_1 a_2 a_3 … a_n
$\mathbf{x_1}$ x^1_1 x^1_2 x^1_3 … x^1_n
$\mathbf{x_2}$ x^2_1 x^2_2 x^2_3 … x^2_n
$\mathbf{x_3}$ x^3_1 x^3_2 x^3_3 … x^3_n
…
$\mathbf{x}_m$ x^m_1 x^m_2 x^m_3 … x^m_n

RESULTS AND DISCUSSION

The data sets used for the study were taken from the UCI Data Repository (http://archive.ics.uci.edu/ml/). Table 1 gives a summary of the ten data sets that were used for all the experiments. Both versions of the stacking algorithm were implemented in Java using the WEKA machine learning suite (http://www.cs.waikato.ac.nz/ml/weka/). The following five learning algorithms were used in both versions of the stacking algorithm: J48 (Quinlan, 1993), Naïve Bayes (John and Langley, 1995), Neural Networks (Kim and Han, 2000), IBk (Aha & Kibler, 1991), and OneR (Holte, 1993). In all experiments the data sets were split 80/20 into a training set and a holdout set as shown in figure 2. In the first set of experiments, *pAtt* = 1 and *pEx* = 0.5 were used. In other words, only instance-based decomposition was used with each sample getting half of the instances from the training data set.

Table 1. Information about the data sets.

Data Set	Attributes	Instances	Attribute Characteristics
Poker	11	25010	real
Letter Recognition	16	20000	integer
Chess	6	28056	categorical
Adult	14	48842	categorical, continuous
Nursery	8	12960	categorical, continuous
Shuttle	9	58000	integer
Mushroom	22	8124	categorical
Pen Digits	16	10992	categorical
Telescope	11	19020	categorical , integer
Block Classification	10	5473	integer , real

Table 2 shows the performance results on the testing set of the two versions of the stacking algorithm along with those of the individual learning algorithms before they are used for creating the meta-classifier. Both versions of the stacking algorithm were run ten times with different random number seeds, and all the results are average of those ten runs. Results (*p* values) of the 1-sided paired *t*-test are also reported to show the significance of the improvement in performance of the standard stacking algorithm over the best learning algorithm, and the improvement in performance of the stacking algorithm using GA as the meta-learner. Significant values (at 0.01 level of significance) are highlighted in bold. Except for the Nursery data set, J48 was the best performing individual learning algorithm on all data sets. The standard stacking algorithm was able to improve the prediction accuracy on five of the ten data sets. The modified stacking algorithm with GA was however able to improve on the performance of the standard stacking algorithm on seven out of the ten sets. The best improvement in performance was on the Chess set, where the modified stacking algorithm was able to improve the prediction accuracy by more than 10% compared to the standard stacking algorithm. The training time is also reported for both versions of the stacking algorithm. On average the modified stacking algorithm takes more time than the standard stacking algorithm since it involves running the GA. Note that both the versions of the stacking algorithm were implemented as sequential algorithms. The training time can be considerably reduced by running the individual learning algorithms in parallel.

Table 2. Predictive Performance Results for pAtt = 1 and pEx = 0.5.

	Individual Learners' Accuracy					Stacking			Stacking with GA		
Data Set	J48	Naïve Bayes	NN	IBk	OneR	Accuracy	Time (sec)	*p*	Accuracy	Time (sec)	*p*
Poker	50.96	50.14	50.29	50.50	50.27	51.25	1235.9	0.460234	52.17	1585.1	0.09515
Letter Recognition	81.12	72.52	75.14	79.45	66.97	90.80	1933.6	***6.83E-12***	94.15	2949.5	***5.98E-06***
Chess	52.33	43.00	45.83	46.52	42.19	57.03	2155.2	***0.000296***	62.75	6970.7	***3.2E-05***
Adult	55.15	54.96	53.98	54.25	52.67	55.37	30993.7	0.078302	55.83	35142.8	***0.010929***
Nursery	94.08	92.11	94.33	94.37	89.73	97.16	945.9	***6.91E-10***	99.52	487.2	***2.29E-06***
Shuttle	95.60	91.85	92.19	92.81	92.40	95.28	726.6	0.22935	99.91	1805.4	***0.019958***
Mushroom	99.95	97.28	98.18	98.63	98.59	99.99	12021.8	***0.007484***	99.96	5219.5	0.148333
Pen Digits	94.00	89.89	91.25	93.17	82.16	98.00	1172.6	***1.57E-10***	99.14	573.3	***2.63E-05***
Telescope	84.08	78.25	80.62	81.19	79.10	84.26	339.1	0.248141	85.86	256.8	***6.39E-08***
Block Classification	96.56	91.99	93.33	93.77	93.67	96.87	167.1	0.075069	96.89	86.9	0.458774

In the second set of experiments, *pAtt* = 0.5 and *pEx* = 0.5 were used. In other words, both instance-based and attribute-based decomposition were used with each sample getting on-average half of the instances containing only half of the attributes from the training data set. Table 3 shows the results for this set of experiments. As before, significant values (at 0.01 level of significance) are highlighted in bold. Note that the performance of all algorithms across the board was worse than in the first set of experiments since they were using only half of all the attributes. J48 was still the best individual algorithm in seven out of the ten sets. The standard stacking algorithm was able to improve the prediction accuracy on four of the ten data sets. The modified stacking algorithm with GA was able to improve on the performance of the standard stacking algorithm on six out of the ten sets. The best improvement in performance was again on the Chess set, where the modified stacking algorithm was able to improve the prediction accuracy by more than 69% compared to the standard stacking algorithm. The training time is also reported for both versions of the stacking algorithm. As before, the modified stacking algorithm takes more time than the standard stacking algorithm since it involves running the GA. The exceptions are the last four data sets for which the modified stacking algorithm is more efficient.

Table 3. Predictive Performance Results for pAtt = 0.5 and pEx = 0.5.

	Individual Learners' Accuracy					Stacking			Stacking with GA		
Data Set	J48	Naïve Bayes	NN	IBk	OneR	Accuracy	Time (sec)	*p*	Accuracy	Time (sec)	*p*
Poker	50.21	50.03	49.93	49.95	49.89	50.05	516.8	0.06231	51.89	848.3	***0.005858***
Letter Recognition	59.68	48.77	51.66	57.53	49.33	79.60	1106.6	***4.26E-05***	84.45	2107.2	0.074477
Chess	26.82	25.03	27.04	27.21	26.27	23.99	1368.5	***1.49E-05***	40.65	2337.8	***6.03E-05***
Adult	53.73	53.44	53.39	53.29	52.73	54.30	9457.2	0.39323	55.82	14365	***0.007819***
Nursery	62.23	61.20	62.39	62.75	61.24	80.86	215	***0.003262***	84.52	375.5	0.263822
Shuttle	96.41	92.40	93.32	94.33	93.97	98.77	661.8	0.100216	95.26	964.3	0.233893
Mushroom	98.99	95.14	96.28	97.04	95.01	99.20	3032.4	0.426326	99.92	1933	0.098741
Pen Digits	83.93	78.50	80.87	84.18	74.62	93.28	539.7	***0.000319***	96.59	356.7	***0.002485***
Telescope	77.14	74.84	75.41	75.64	74.27	78.99	204.1	0.217304	82.44	174.6	***0.007871***
Block Classification	95.10	90.39	91.30	91.97	92.05	95.13	92.2	0.474408	95.93	54.8	***0.013562***

In both sets of experiments, the modified stacking algorithm was able to improve the performance of the standard stacking algorithm in majority of the data sets tested. This shows the potential of using a genetic algorithm to improve the performance of ensemble learning algorithms. Note that there was no attempt to tailor the ensemble learning algorithm for a given data set. One could possibly improve the performance of this modified stacking algorithm independently for each data set even further by fine tuning several parameters such as, the number and type of individual learning algorithms, the parameters of each of these individual algorithms, the value of *pAtt* and *pEx*, and the parameters of the genetic algorithm.

CONCLUSION

In this paper we presented a modified version of the standard stacking ensemble algorithm that uses a genetic algorithm to create an ensemble. We also tested two data decomposition techniques to distribute the data over the individual learning algorithms in the ensemble. We tested the GA-based stacking algorithm on ten data sets from the UCI Data Repository and showed the improvement in performance over the individual learning algorithms as well as over the standard stacking algorithm. We are currently also working on testing the robustness of the algorithm in the presence of noise.

REFERENCES

Aci, M., Inam, C., & Avci, M. (2010). A hybrid classification method of k nearest neighbors, Bayesian methods and genetic algorithm. *Expert Systems with Applications*, 37(7), 5061-5067.

Agustin-Blas, L., Salcedo-Sanz, S., Jimenez-Fernandez, S., Carro-Calvo, L., Del-Ser, J., & Portilla-Figueras, J. (2012). A new grouping genetic algorithm for clustering problems. *Expert Systems with Applications*, 39(10), 9695-9703.

Aha, D., & Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, 6, 37-66.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123–140.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2), 139–158

Dietterich, T. (2002). Ensemble learning, in The Handbook of Brain Theory and Neural Networks, 2nd ed., M. Arbib, Ed., Cambridge MA: MIT Press.

Dzeroski, S., & Zenko, B. (2004). Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 255–273.

Freitas, A. (2002). Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new Boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 148-156

Goldberg, D. (1989). Genetic Algorithms in Search, Optimization & Machine Learning, Addison Wesley.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11, 63-91.

IBM. (2012). Bringing Big Data to the Enterprise: http://www-01.ibm.com/software/data/bigdata/

John, G., & Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. *Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo*, 338-345.

Kim, K., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19(2), 125-132.

Opitz ,D., & Maclin, R. (1999). Popular ensemble methods: an empirical study, *Journal of Artificial Intelligence Research*, 11, 169-198.

Pal, M. (2007). Ensemble learning with decision tree for remote sensing classification. *Proceedings of World Academy of Science, Engineering and Technology*, 26, 735–737.

Quinlan, R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

Sexton, R. S, Sriram, R. S., & Etheridge, H. (2003). Improving decision effectiveness of artificial neural networks: A modified genetic algorithm approach. *Decision Sciences,* 34(3), 421-442.

Sikora, R., & Piramuthu, S. (2005). Efficient Genetic Algorithm based Data Mining using Feature Selection with Hausdorff Distance. *Information Technology and Management*, 6(4), 315-331.

UCI Machine Learning Repository, Center for Machine Learning and Intelligent Systems, http://archive.ics.uci.edu/ml/

Weka-3: Data Mining with Open Source Machine Learning Software in Java, http://www.cs.waikato.ac.nz/ml/weka/

Zeng, L., Li, L., Duan, L., Lu, K., Shi, Z., Wang, M., Wu, W., & Luo, P. (2012). Distributed data mining: a survey. *Information Technology and Management*, 13(4), 403-409.

This Page Intentionally Left Blank