2015 International Conference on Computational Science and Computational Intelligence

Model Based Sampling Fitting an ensemble of models into a single model

Tony Lindgren
Department of Computer and Systems Sciences
Stockholm University
Borgarfjordsgatan 12
164 40, Kista, Sweden
Email: tony@dsv.su.se

***Abstract*—Large ensembles of classifiers usually outperform single classifiers. Unfortunately ensembles have two major drawbacks compared to single classifier; interpretability and classifications times. Using the Combined Multiple Models (CMM) framework for compressing an ensemble of classifiers into a single classifier the problems associated with ensembles can be avoided while retaining almost similar classification power as that of the original ensemble. One open question when using CMM concerns how to generate values that constitute the synthetic example. In this paper we present a novel method for generating synthetic examples by utilizing the structure of the ensemble. This novel method is compared with other methods for generating synthetic examples using the CMM framework. From the comparison it is concluded that the novel method outperform the other methods.**

Keywords*—Machine learning algorithms, Supervised learning, Sampling methods, Approximation algorithms.

I. Introduction

The usage of ensemble methods in the field of machine learning has in numerous studies proven to outperform single models and the reasons for their strengths are well known, see for example [1], [2]. When comparing single models with ensemble of models the former has two major attractive advantages against the latter. The first advantage is the comprehensibility of the model and the second advantage is resource efficiency which can be split into two parts (which go hand in hand) model size and classification speed.

In paper [3] P. Domingos introduces an algorithm called Combined Multiple Models (CMM) for creating single models from ensembles of models such that the predictive performance is not affected to any major extent while making the model comprehensible as it consists of a single decision tree. The focus of this work is especially on the comprehensible part as a motivation of why one should transform an ensemble of models to a single model.

The main idea of CMM is to generate synthetic examples which in turn are labeled by the ensemble. These labeled examples are then used, together with the original training data, to create a new single model. The function of the ensemble is in this way transferred from the ensemble via the labeled examples to the single model.

More recently C. Bucila et. al. in their paper [4], presented another method based on CMM which address the other major advantage of single models, i.e. their small memory footprint and classifying speed by training an Artificial Neural Network (ANN) with synthetic examples labeled by an ensemble of models. Their work aimed at making it possible to import and use the ANN on weak hardware, like PDAs while keeping the predictive performance and ensuring quick response times for the classification. In their paper they introduce a novel method for creating synthetic examples which they call MUNGE which they compare to Random Uniform Sampling (RUS) and Naive Bayes Estimation (NBE) [5]. In their experiments MUNGE outperformed the other two methods.

In this paper we introduce a non-parametric method for generating synthetic data which utilizes the structural dependencies that the ensemble of models consists of. The rest of the paper is structured as follows: In the next section we will present our method for generating synthetic data together with a quick recapitulation of the MUNGE method and the Random Uniform Sampling method. We then describe our experimental setup where we compare the three methods for generating synthetic examples and present the results from the experimental evaluation. Finally we end the paper with a discussion and conclusions and give pointers to future work.

II. Methods for synthetic example generation

The overall goal for why it is interested to generate synthetic examples is that we want to use these examples as mediators when moving from an ensemble of models to a single model. Hence we want to capture the predictive performance of the ensemble of models in a single model and one way of doing this is to generate synthetic examples which then are labeled by the ensemble of models and later on used to train a single model together with the original training data. This follows the CMM methodology as mentioned earlier.

But how does one go about in selecting values for the synthetic data in such a way that it captures the important aspects of the ensemble. Clearly the best values would be real unlabeled examples coming from the same data source as the labeled training data. Unfortunately it is not always possible to extract new unlabeled examples that can be used for training purposes and then a method for synthetic data selection/generation is needed. When generating synthetic values for the synthetic examples it is important that this is done in such a way that these values would mimic the true distribution from which the original labeled examples is drawn from.

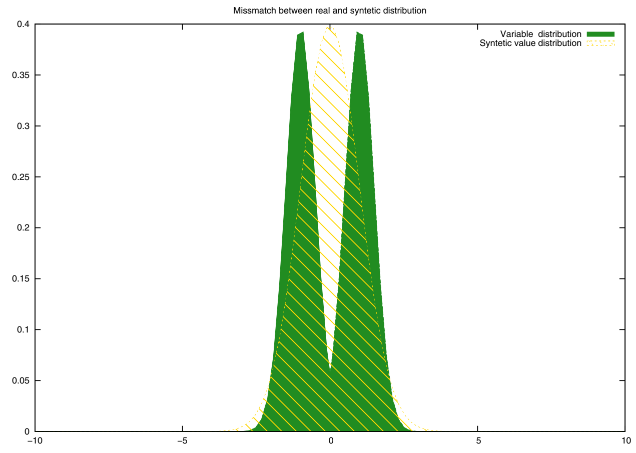


Fig. 1: Showing the mismatch between a variables true distributions and distribution used for generating synthetic examples

Assuming normal distribution this could be done by capturing the mean and the standard deviation for each attribute and using the values as starting points for generating new values according to the probability of the normal distribution. This method has of course the problem that if the distribution differs from the assumed one we have a problem, e.g. if we have an continuous attribute and the training examples create two peaks on the number line we would not capture this very well using the normal distribution.

See figure 1 for an illustration of this problem. This could of course be handled by dividing up the number line, using other distributions, usage of clustering techniques etc.

Another issue to consider when generating synthetic examples is dependencies between values, for example having examples containing *Sex* and *Number of pregnancies* would suggest that it would be impossible to generate synthetic examples with males which have been pregnant more than zero times. This could be tackled by the use of background knowledge of the domain that could capture the most obvious constraints which must hold for a synthetic example.

After getting the grasp with some of the problem we are facing when trying to generating synthetic examples we will now present the different approaches that we will compare for generating synthetic examples, including our novel method which we call *Model Based Sampling*.

A. Random Uniform Sampling

Random Uniform Sampling (RUS) can be considered as a baseline method which we will test the more sophisticated methods against. The method is simple both to understand and implement but it has downsides as it will most probably generate synthetic example values that lies outside the region of interest. This is the case as if the "true" distribution for some attribute has one or more regions where the value(s) are concentrated around RUS will generated values which will be spread with equal probability over the whole range of the attributes possible values.

The method works as follows, for one attribute find the minimum and maximum value. Use these values as boundaries and randomly select a value that fall in-between the boundaries. Do this for all the attributes that constitutes an example except the class label and for as many times as one needs new synthetic examples. The process is described more formal in the Algorithm 1.

The algorithm takes as input training examples and the number of synthetic examples to generate ($SampleSize$). It then calls the function **GetMinMax** which takes the training examples as arguments. This function loops through each attribute and each example in the training examples and collects the minimum and maximum values for each attribute and returns a list containing the maximum and minimum values for each attribute. The function **GenExs** is then called with the list of minimum and maximum values for each attribute together with the sample size, which denotes the number of synthetic examples to generate. In **GenExs** the maximum and minimum attribute values are used as boundaries for a random number generator.

Algorithm 1 Random Uniform Sampling

```
Inputs: $TrainingEx, SampleSize$
Outputs: $SynteticExamples$

$AttVals \leftarrow \text{GETMINMAX}(TrainingEx)$
$SynteticEx \leftarrow \text{GENEXS}(SampleSize, AttVals)$
return $SynteticEx$

function GENEX($SampleSize, AttVals$)
    $SynteticExs \leftarrow \emptyset$
    while $0 < SampleSize$ do
        $SynteticEx \leftarrow \emptyset$
        for each attribute $a_{min,max} \in AttVals$ do
            $Val_a \leftarrow \text{GETRNDVAL}(a_{min,max})$
            $SynteticEx \leftarrow Val_a$
        end for
        $SynteticExs \leftarrow SynteticEx \cup SynteticExs$
        $SampleSize \leftarrow SampleSize - 1$
    end while
end function
```

B. MUNGE

For a more thorough examination of the MUNGE method see [4]. MUNGE starts from the original training set and for each example finds its closest neighbor, using Euclidian distance for continuous attributes. All attributes are scaled to the interval [0, 1]. The algorithm takes two parameters, one probability parameter *Prob* and a local variance parameters S. New examples are generated by interchange of values between an example e and its closest neighbor e', each attribute e_a are interchanged with the probability *Prob*.

The interchange itself works as follows: a new value e_a is generated from the normal distribution, with the standard deviation $sd = |e_a - e'_a|/s$ and e'_a as mean. The value for e'_a is generated in similar fashion but with e_a as it's mean. The algorithm is described more formally in Algorithm 2[1] MUNGE takes as input the probability $Prob$ and the local variance S as mentioned before, training examples and $SampleSize$, which denotes the number of synthetic examples to generate.

Algorithm 2 MUNGE

```
Inputs: $TrainingEx, SampleSize, Prob, S$
Outputs: $SynteticExamples$

D $\leftarrow \emptyset$
**while** $0 < SampleSize/2$ **do**
    **for** each example $e \in TrainingEx$ **do**
        $e' \leftarrow$ the closest example of $e$ from $TrainingEx$
        **for** each attribute $a \in e$ **do**
            with probability $Prob$: $e_a \leftarrow norm(e'_a, sd)$ and
            $e'_a \leftarrow norm(e_a, sd)$, where $sd \leftarrow |e_a - e'_a|/s$,
            and, $norm(a, b)$ is the random value taken from
            the normal distribution with mean $a$
            and standard deviation $b$.
        **end for**
    **end for**
    $D \leftarrow D \cup$ new $e \cup$ new $e'$
**end while**
**return** $D$
```

C. Model Based Sampling

In contrast to the RUS and MUNGE Model based Sampling, utilizes the ensemble of models not just for setting the class label of the generated synthetic examples but also for generating the examples values. Having an ensemble consisting of decision trees it ranks all leaf nodes in the trees according to how many examples from the training set it covers. The total number of example covered $totNum$ in the leaf nodes is then used as a basis for a probability mapping. By randomly selecting a number within 0 to $totNum$ on can then select a leaf node with a probability proportional to the chance that a random example (drawn from the same source as the training examples) would fall into this leaf. Using the conditions for the leaf, i.e. the path from the leaf to the root node, we establish the boundaries that must be preserved by the synthetic example values. Using attributes that are present in the conditions of the leaf together with the training examples that fall into the leaf, we can calculate for each attribute its mean and standard deviation. We the use these values to sample from the normal distribution with the mean and standard deviation as parameters and also checking that the generated values do not violate the conditions of the leaf node. The generated attribute values are then removed from the list of attributes that needs values for a particular synthetic example.

We then start the algorithm over again, but firstly we filter out all leaf nodes that have conditions that are contradictory to the attribute values we just assigned to our synthetic example. Hence only keep leaf nodes that are compliant with the values of the synthetic example. Then the previous step is repeated until all attributes has a value, and the example is complete and ready to be label by the ensemble.

There are is a few special cases that must be pointed out; it can for example be the case that some attributes are not used by any leaf. This special case are handled and filtered out before we run the basic algorithm, the attributes (if there are any) that do not have any leaf nodes are assigning values based on all training data's mean and standard deviation for these attributes. It can also be the case that after we filter out contradictory leaf nodes we are left with an empty list, we the handle this by generating values for these attributes based on all training instances mean and standard deviation for these attributes. The algorithm is shown in Algorithm 3.

Algorithm 3 Model Based Sampling

```
Inputs: $TrainingEx, SampleSize, Trees$
Outputs: $SynteticExs$

**while** $0 < SampleSize$ **do**
    $ex \in TrainingEx$
    $Atts \leftarrow$ all attributes in $ex$
    **while** $Atts! = \emptyset$ **do**
        $notUsedAtt \leftarrow$ NOTUSEDATTS($Trees$)
        **if** $notUsedAtts! = \emptyset$ **then**
            calculate mean and std deviation for
            attributes $\in notUsedAtts$, $Atts_m, Atts_{std}$
            $AttVals \leftarrow$ NORM($Atts_m, Atts_{std}$)
            $Atts \leftarrow Atts \cap notUsedAtts$
        **end if**
        $LeafN \leftarrow$ OKAYLEAFNODES($Trees, Atts$)
        **if** $LeafN == \emptyset$ **then**
            calculate mean and std deviation for
            remaining attributes, $Atts_m, Atts_{std}$
            $AttVals \leftarrow$ NORM($Atts_m, Atts_{std}$)
            $Atts \leftarrow \emptyset$
            $SampleSize \leftarrow SampleSize - 1$
        **else**
            $SortLeafN, TotNum \leftarrow$ SLEAFS($LeafN$)
            $rndVal \leftarrow$ RND($0, TotNum$)
            $leafN \leftarrow$ GETLEAFN($rndVal, SortLeafN$)
            $AttVals \leftarrow$ NCOND($mean, stddev, leafN$)
            $Atts \leftarrow Atts \cap AttVals$
        **end if**
    **end while**
    $SampleSize \leftarrow SampleSize - 1$
    $SynteticEx \leftarrow AttVals$
    $SynteticExs \leftarrow SynteicExs \cup SynteticEx$
**end while**
```

Model Based Sampling takes as input the number of synthetic examples to generate ($SampleSize$) and the forest of decision trees. The outer while loop checks with $SampleSize$ to see if more examples are to be generated, if so attributes $Atts$ are initialized. The inner while loop check if there are still attributes that needs values assigned to them for the current synthetic example, if so the function **NotUsedAtts** checks the conditions of the forest and returns the attributes not used in conditions. If such attributes exist their values are generated using all training data to compute the attribute(s) mean and standard deviation as input to the normal distribution, other-

[1]We only present the MUNGE's method for usage with continuous values as this is what we use in this paper. We have also changed the algorithm slightly, it previously generated multiples of the original training data, and we instead use a parameter the set the number of examples to generate.

wise the function **OkayLeafNodes** filter out leaf nodes of the forest which are compatible with the current attribute values of the example (if any are present).

If no leaf nodes are compatible with current assigned attribute values for the remaining attributes are generated using the normal distribution with all training examples used for calculating each attribute(s) mean and standard deviation. $SampleSize$ is then reduce by 1 and $Atts$ is set to the empty set. Otherwise the leaf nodes are sorted so that they "occupy" a space proportional to the number of (training) examples that they cover. This is done by the function **SLeafs** which also return the accumulated number of examples in the leafs (note that this figure probably is much higher than that of the number of training instances, as some examples are cover by multiple leafs). A random number generator, **Rnd** generate a number between 0 and TotNum. The random number $rndVal$ is then used by the function **GetLeafN** to extract the leaf node pointed out by the random value. The function **NCond** generate values for the attributes that the leaf node has as conditions (on its path from the leaf to the root node) for these attributes the function calculates the mean, standard deviation for the examples in the leaf node and uses them for generating new values it also checks that the conditions of the leaf node is not violated. The generated attributes are then removed from $AttVals$ and finally if $AttVals$ is empty $SampleSize$ is reduced by one and the attribute values are added to the synthetic example which in turn is added to the set of synthetic examples.

III. Experimental evaluation

Evaluation of the methods is done on 14 data sets all taken from the UCI repository [6]. All data sets contain only attributes with numerical data. In the case of MUNGES two parameters, the probability and local variance parameter, there were no clues of what would be sensible / good value to use. In the paper [4] they did not present which values they used in their experiment. This is not ideal and we had to come up with sensible values for these parameters.

A upper bound of 0.5 for the probability parameters is easily motivated as if it is exceeded more than half of the values would come from another example, this value was in our case set to 0.25, e.g. if an example contains 4 attributes and one class label this would on average change 1 attribute of the synthetic examples. The local variance is harder to reason about to find a sensible value as it in my opinion needs to be set dynamically for each attribute of a data set, but that is not correct according to the MUNGE algorithm where it is an input to the algorithm and constant over all attributes. How to choose a good value for this parameter is a mystery so we resorted to select a value ad-hoc. The value was set to 4 with no further motivation.

A. Experimental setup

The three methods where evaluated in a 10-fold cross validation scheme. Results are also shown for a single tree and the original ensembles result. The ensemble size where set to 100, i.e. a forest consisting of 100 decision trees. The ensemble strategy was that of bagging [7] thus creating a new training set for each of the 100 trees by sampling examples with replacement.

Two data sets contained missing values in the case of breast cancer Wisconsin 16 missing values where replaced by the value 1, in the case of Cleveland heart disease all missing values were replace by 0.0. Different synthetic sample sizes where used for all three methods, the sizes were: 100, 250, 500, 750, 1000, 2500, 5000, 7500 and 10000. So the methods generated this amount of synthetic data which then was labeled by the forest and used together with the original training data to build a single tree.

From the experiments the average accuracy and average tree size over the 10 folds are measured. The datasets ranges from size of 150 to 7494 instances, the number of attributes ranges from 3 to 34 attributes excluding class label. The algorithms where implemented with Sicstus Prolog, and are available to downloaded from http://dsv.su.se/~tony/programs.html together with the data sets.

B. Experimental results

Given any dataset we expect that the accuracy of single tree would be lower than that of the ensemble of 100 trees. This is also true in the general case. It is also expected that the single tree and the ensemble would serve as boundaries for the sampling methods which accuracies would fall in between the accuracies of the single tree and the ensemble. This is also true for the general case but as we will see when looking at the results sometimes the sampling methods actually outperform the ensemble, but there is also one instance where the sampling methods perform worse than expected and fall below the performance of the single model.

In table I the average accuracies over the ten-folds is shown for the ensemble and the single tree, together with the average rule size, i.e. number of paths from leaf node to root. For sampling methods the average value over all sample sizes is shown together with the worst-best accuracies, shown within parenthesis. Their rule sizes in the table are also averaged over all sample sizes. The best average result for a sampling method is marked in bold font. Here we see that when comparing the synthetic sampling methods, MBS wins over the other methods in 8 out of 14 domains, the same figure for RUS is 4 out of 14 domains and for MUNGE 2 out of 14 domains. This seems to indicate that MBS outperforms the other methods. If one looks at the sizes of the trees constructed by the synthetic sampling methods, there is no clear pattern or difference between the methods.

Following the significance test procedure of [8][2], we first apply the Friedman test, the ranking is shown in the table in the row with *Friedman rank 1* in the first column. As expected the Ensemble method are ranked best then comes MBS and RUS, surprisingly MUNGE is ranked after Single Tree. The p-value of the test is: 0.00004321, which indicate that we indeed have statistical significant differences between the methods. As we are interested in the differences between the sampling methods we exclude the ensemble method and the single tree and re-run the Friedman test.

The result is shown in the row marked *Friedman rank 2* the P-value is: 0.03019738 which still denote a significant

[2]The statistical test are performed with the software that can be found at: http://sci2s.ugr.es/keel/multipleTest.zip.

TABLE I: Average result table

Data set	Ensemble	Single tree	Random Uniform Samp.	Model Based Samp.	MUNGE
Breast c. w.	95.3, 1020.3	94.7, 13.9	91.0 (90.7-91.8), 20.7	**94.6 (93.7-95.1), 35.7**	94.1 (93.3-94.9), 24.5
BUPA liver d.	71.9, 3887.9	62.6, 55.6	59.5 (56.2-62.3), 129.2	**66.8 (61.4-70.1), 136.6**	62.1 (60.3-64.9), 210.6
Cleveland h.	56.0, 4140.2	51.0, 55.0	52.1 (48.3-54.3), 266.1	**54.7 (51.7-56.3), 230.4**	50.2 (47.0-53.0), 88.7
Climate model	93.7, 1078.6	93.3, 14.4	**94.1 (93.1-94.8), 28.6**	93.5 (92.6-94.6), 27.9	93.6 (92.2-94.6), 19.2
Glass id.	94.5, 369.2	93.6, 4.8	94.0 (91.8-95.4), 17.9	**94.1 (92.7-95.4), 5.1**	92.6 (90.9-93.6), 5.0
Haberman	68.0, 4350.8	65.4, 65.3	68.3 (62.4-70.6), 143.0	**70.2 (67.3-73.5), 149.9**	68.0 (58.8-72.2), 168.0
Image seg.	89.5, 1219.3	85.2, 13.5	86.3 (84.7-87.6), 55.8	87.4 (85.6-90), 72.2	**89.1 (87.6-90.9), 16.8**
Ionosphere	74.0, 1028.8	66.3, 14.9	75.8 (70.0-82.3), 73.4	**76.5 (73.7-82.3), 122.7**	73.3 (70.0-81.1), 83.8
Iris	96.7, 436.4	95.3, 4.8	**95.7 (95.3-96.0), 6.0**	95.4 (94.6-96.0), 7.4	95.0 (93.3-96.7), 6.1
Thyroid dis.	94.4, 475.7	93.0, 5.6	87.2 (81.9-94), 5.2	92.9 (92.1-93.5), 9.4	**93.5 (93.0-94.4), 5.9**
Pen digitis	96.1, 9122.0	94.4, 100.6	**93.7 (92.6-94.6), 193.2**	93.6 (92.7-94.2), 208.1	92.4 (91.0-94.4), 262.6
P. i. diabetes	76.3, 6064.6	69.0, 86.1	71.8 (66.0-73.9), 175.0	**74.0 (71.1-77.1), 179.4**	70.5 (66.9-73.2), 190.9
Waveform n.	82.9, 15465.8	77.4, 253.3	77.7 (77.3-78.3), 313.0	**77.8 (77.2-78.6), 315.3**	77.3 (76.8-77.8), 227.0
Wine	96.1, 506.1	93.8, 5.1	**93.9 (93.3-94.9), 17.0**	93.5 (91.6-94.4), 11.9	93.4 (91.6-95.5), 7.9
Friedman rank 1	1.393	3.857	3.071	2.643	4.036
P value	0.00004321				
Friedman rank 2	-	-	1.999	1.499	2.5
P value	0.03019738				

difference between the methods, although not as strong as before, according to the probability threshold of 5 percent. To further investigate the statistical differences between the synthetic sampling methods we conducted paired tests. In table II the first column show which pair of sampling methods are compared in that row, the second row show the unadjusted P-value, the following rows show the adjusted P-values for Nemenyi's, Holm's, Shaffer's and Bergmann's procedure respectively.

From the table it is clear that MBS differs significantly from MUNGE, while the rest of the methods do not differ significantly.

TABLE II: Adjusted p-values

hypothesis	unadjusted p	p_{Neme}	p_{Holm}	p_{Shaf}	p_{Berg}
MBS vs MUNGE	0.00815097	0.0244529	0.0244529	0.02445291	0.02445291
RUS vs MUNGE	0.18587673	0.5576302	0.3717535	0.18587673	0.18587673
RUS vs MBS	0.18587673	0.5576302	0.3717535	0.18587673	0.18587673

In figure 2 four domains and the corresponding experimental results are shown, on the horizontal axis the number of synthetic examples created are denoted and on the vertical axis the accuracies for the different methods are plotted. The accuracy of the single tree and ensemble methods are displayed as a vertical line in green respectively red, as their accuracy is not affected by the synthetic sample size. In all subfigures the RUS accuracy is plotted using a blue line, MBS using a pink line and MUNGE magenta colored line. Subfigure (a) and (b) display a typical, and predicted, pattern of the methods where the accuracies is typically in between the accuracy bound of the single tree and the ensemble. The trend is also that the accuracy increases with larger sample sizes, even though this is not monotonic. The subfigure (a) shows the Cleveland heart disease examples set and subfigure (b) the bupa (liver disorder dataset).

In subfigure (c) which regards the Ionosphere data set the results are quite strange and not what we expected, the MBS method almost outperforms the ensemble methods on all sample sizes. Even more impressive is the performance from RUS methods on sample sizes from 2500 and upwards. Where it reaches an accuracy of just above 82 percent compared with the ensemble accuracy of 74 percent. MUNGE perform well initially but then drops in accuracy but perform as expected, i.e. between ensemble and single tree. One possible explanation of why RUS perform so well could be that is samples from potential values more freely than both MBS and MUNGE, which could be beneficial in this domain.

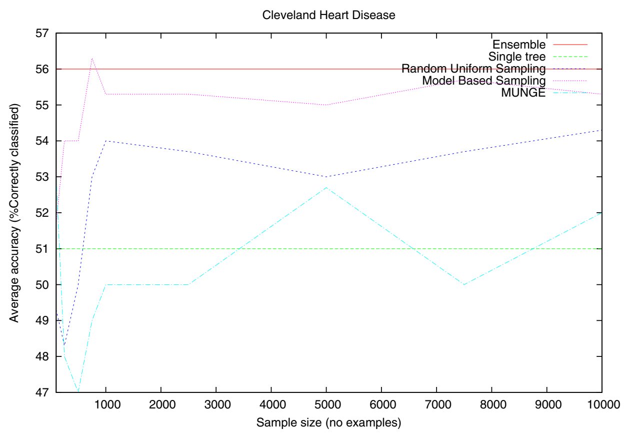
Finally in subfigure (d) the roles are switched there all sampled methods perform worse than expected, i.e. the all fall below that of the performance of the single tree. Why the methods fail on this data set is unclear at the moment and this needs further investigation. Given limited space not all figures from the experiment could be included in the paper.

IV. Discussion and conclusions

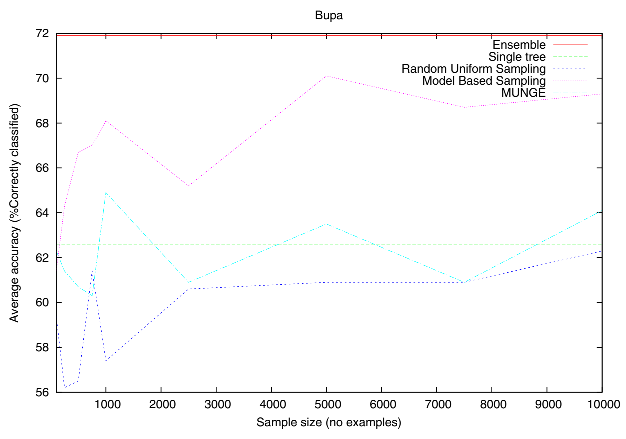
We have introduced a novel method for generating synthetic examples, using the CMM approach these examples are labeled by an ensemble of classifiers and then used for constructing a single tree. The method was expected to have a performance that would lay in between that of the original ensemble but higher than a single tree constructed from the original data. This notion was confirmed by the Friedman ranking and of the methods, except for the MUNGE method which was ranked last ad hence after single tree method. This could maybe be explained by wrong parameter choice for the method, and if this is the case, the *local variance* is probably the parameter that needs attention.

The fact that RUS and MBS do not need to have parameters set is another major advantage for both these methods compared MUNGE. MBS do perform better than RUS but the difference is not statistically significant. Albeit not in focus in this study the time complexity for MBS much higher than for RUS and MUNGE, as trees in the ensemble needs to be traversed repeatedly for each generated synthetic example. But this is of course a single effort and after it is done the model can be used repeatedly so the extra complexity could be worth the effort, in order to achieve better classification performance.

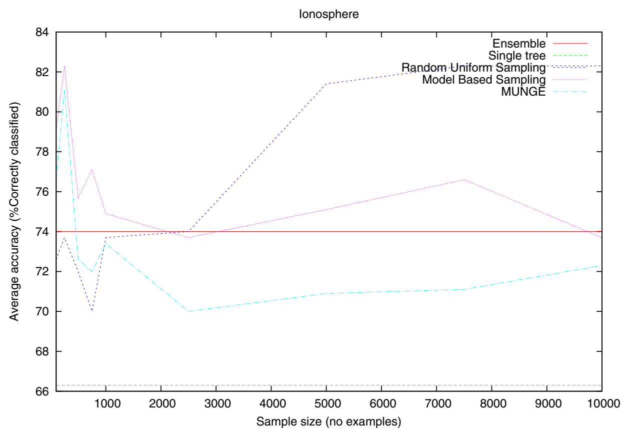
The MBS results also (through the empirical results) support the notion that is important to keep track of the distributions in the leaf nodes of the ensemble when constructing synthetic examples, as P. Domingos pointed out in his paper [3].



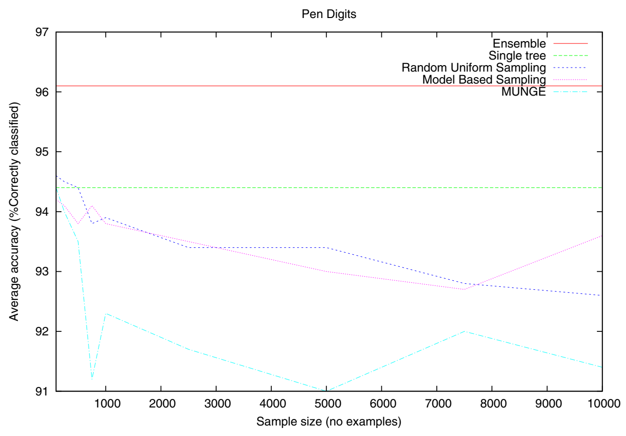
(a) Plotted performance for sample methods on Cleveland h.



(b) Plotted performance for sample methods on Bupa



(c) Plotted performance for sample methods on Ionosphere



(d) Plotted performance for sample methods on Pen Digits

Fig. 2: Experimental results from four example domains

A. *Future work*

Hopefully the MBS method can be further improved; one issue that needs attention is the time complexity of the method. Ways for addressing this is an open venue for research.

Investigating why MUNGE performed so badly is something that needs to be done. Can/should the local variance be set (dynamically) for each data set? A possible solution is to pre-process each data set and compute the local variance for each attribute. These values can then be used as input to the algorithm for each attribute of a particular data set.

V. Acknowledgments

This work has been funded by Scania CV AB and the Vinnova program for Strategic Vehicle Research and Innovation (FFI)-Transport Efficiency.

References

[1] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[3] P. Domingos, "Knowledge acquisition from examples via multiple models," in *In Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 98–106.

[4] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, 2006, pp. 535–541. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150464

[5] D. Lowd, "Naive bayes models for probability estimation," in *Proceedings of the Twentysecond International Conference on Machine Learning*. ACM Press, 2005, pp. 529–536.

[6] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[7] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996. [Online]. Available: http://dx.doi.org/10.1023/A:1018054314350

[8] S. Garca and F. Herrera, "An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons," *Journal of Machine Learning Research*, vol. 9, pp. 2677–2694, 2008.