1. Algorithm Name

Boosting-Based pruning (BB)

2. Reference

Martínez-Muñoz, G., & Suárez, A. (2007). Using boosting to prune bagging ensembles. *Pattern Recognition Letters*, *28*(1), 156-165

3. Motivation for the algorithm (or which problems it tries to solve?)

This method intends to use ensemble pruning to reduce storage and running time. “Numerous experimental studies show that pooling the decisions of classifiers in an ensemble generally improves the generalization performance of weak learners. An important shortcoming of ensemble methods is the loss of classification speed and the growth in storage needs with increasing numbers of inducers.
Ensemble pruning reduces the storage needs, speeds up the classification process and has the potential of improving the classification accuracy of the original ensembles.”

4. Short Description:

This method selects a subset of learners from a pool of classifiers previously generated by bagging. The ensemble grows by selecting at each step the classifier that minimizes a weighted error on the selection set. The process ends when a predefined size for the final pruned ensemble is reached.
“This is ensemble pruning method that uses the weighted training error defined in boosting to determine the order in which classifiers are aggregated in an initially randomly ordered bagging ensemble. The algorithm proceeds iteratively by updating the training example weights as in boosting: The weights of training examples correctly (incorrectly) classified by the last classifier incorporated into the ensemble are decreased (increased) according the AdaBoost prescription Freund and Schapire (1995). The classifier that minimizes the weighted training error is then incorporated into the ensemble.”

5. Pseudo-Code

Boosting-Based pruning - Building the ensemble

Input:
$S = (< x_1, y_1 >, ..., < x_m, y_m >)$ training set composed of m instances
T- number of classifiers
U- classifiers to select

1. $for\ t = 1\ to\ T\ \{$
2. $\quad L_{bt} = BootstrapSample(S)$
3. $\quad I_t = TrainClassifier(L_{bt})$
4. $\quad Add\ I_t\ to\ pool\ C$
5. $\}$
6. $set\ all\ instance\ weigths\ w\ to\ 1/m$
7. $for\ u = 1\ to\ U\ \{$
8. $\quad //Gets\ the\ learner\ with\ lowest\ weighted\ error$
9. $\quad M_u = SelectBest(C, S, w)$
10. $\quad \in_u = WeightedError(M_u, S, w)$
11. $\quad if\ (\in_u > 0.5)\{$
12. $\quad\quad reset\ all\ instance\ weights\ w\ to\ \frac{1}{m}$
13. $\quad\quad GOTO\ 8$
14. $\quad \}$
15. $\quad Add\ M_u\ to\ pool\ M$
16. $\quad Extract\ M_u\ \ from\ pool\ C$
17. $\quad for\ j = 1\ to\ m\ \{$
18. $\quad\quad if\ (M_u(x_j) \neq y_j)\ then\ w_j = \frac{w_j}{2\in_u}$
19. $\quad\quad else\ w_j = \frac{w_j}{2(1-\in_u)}$
20. $\quad \}$
21. $\}$
22. $return\ M$

Boosting-Based pruning - Classify an instance
Input: x – an instance needed to be labeled

1. return $\sum_{u:M_u(x)=y} 1$

Figure 1: The Boosting-Based pruning Pseudo Code

6. Algorithm Explanation:

The algorithm gets as input the following: Training set, number of classifiers and percentage of classifiers to select. According to the paper, we had two options for the pruning rule: the number of classifiers or aggregate classifiers until the first classifier with a weighted training error above 0.5 is found. We chose the first option since it consistently produces better results in classification problems.

In lines 1-5 we generate a pool of classifiers by bagging ensemble of is constructed from the training set. Instead of bagging, other parallel ensemble building methods can in principle be used.
In line 6 we initialize all instances with the same weight.
In lines 8 we select from the pool of classifiers generated by bagging the best classifier, i.e. with the lowest weighted training error.
In lines 9-13 we calculate the weighted training error in order to avoid early stopping. if no classifier has a weighted training error below 50%, the weights of the examples are reset to the same weight.
In lines 14-20 we update the weights: The weights of training examples correctly classified by the last classifier incorporated into the ensemble are decreased and vice versa. We didn't weighted the voting in the ordered bagging ensembles since there is no significant variation between weighted to unweighted voting.
The process continues until the desired amount of classifiers is reached.
The final decision is performed with either unweighted or weighted voting of the selected classifiers.

7. Illustration

In this section we illustrate the running of Boosting Based Pruning algorithm using the following settings T=10 and U=50. The decision Tree algorithm is chosen to be the base inducer. A simplified version of the **elusage** dataset (Table 1) is used as the training set. Then, 10 decision trees are trained over bootstrap sample of the training set. The classes distribution of each learner has shown in Table 2.
Initially all instances have the same weight. Now, the algorithm selected the best learner with the minimal weighted error on the original training set $\epsilon_t \approx 0.02$.
Thus, the instances are reweighted according to procedure indicated in lines 18-19. The new instances weights are presented in the third column in Table 3. Note that the weights of the correctly classified instances decreased to 0.00 while the weights of the incorrectly classified instances increased to 0.25. The second learner is chosen with $\epsilon_t \approx 0.02$. The updated weights are presented in the fourth column in Table 3. The process continues until the 5 classifiers is reached according to U (50*10/100). The weighted errors in each iteration are presented in figure 2.

Table 1: Initial dataset

average_temperature	month	binaryClass
73	8	P
67	9	P
57	10	P
43	11	N
26	12	N
41	1	N
38	2	N
46	3	N
54	4	N
60	5	P
71	6	P
75	7	P
74	8	P
66	9	P
61	10	P
49	11	N
41	12	N
35	1	N
41	2	N
42	3	P
56	4	P
69	5	P
73	6	P
77	7	P
74	8	P
66	9	P
56	10	P
47	11	P
38	12	N
34	1	N
37	2	N
39	3	N
51	4	N
60	5	P
70	6	P
73	7	P
71	8	P
66	9	P
54	10	N
45	11	N
36	12	N
34	1	N
32	2	N
39	3	N
55	4	P
64	5	P
72	6	P
79	7	P
75	8	P
65	9	P
54	10	P
48	11	P
35	12	N
24	1	N
32	2	N

Table 2: Classes Distribution

	learner 1	learner 2	learner 3	learner 4	learner 5	learner 6	learner 7	learner 8	learner 9	learner 10
P	30	34	30	31	30	32	29	37	30	28
N	25	21	25	24	25	23	26	18	25	27

Table 3: Weights of instances in Boosting Based Pruning iterations.

Sample	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
0	0.01	0.00	0.00	0.00	0.00
1	0.01	0.00	0.00	0.00	0.00
2	0.01	0.00	0.00	0.00	0.00
3	0.01	0.00	0.01	0.01	0.01
4	0.01	0.00	0.00	0.00	0.00
5	0.01	0.00	0.00	0.00	0.00
6	0.01	0.00	0.00	0.00	0.00
7	0.01	0.00	0.00	0.00	0.00
8	0.01	0.00	0.00	0.00	0.00
9	0.01	0.00	0.00	0.00	0.00
10	0.01	0.00	0.00	0.00	0.00
11	0.01	0.00	0.00	0.00	0.00
12	0.01	0.00	0.00	0.00	0.00
13	0.01	0.00	0.00	0.00	0.00
14	0.01	0.00	0.00	0.00	0.00
15	0.01	0.25	0.17	0.10	0.07
16	0.01	0.00	0.00	0.00	0.00
17	0.01	0.00	0.00	0.00	0.00
18	0.01	0.00	0.00	0.00	0.00
19	0.01	0.00	0.00	0.01	0.01
20	0.01	0.00	0.00	0.00	0.00
21	0.01	0.00	0.00	0.00	0.00
22	0.01	0.00	0.00	0.00	0.00
23	0.01	0.00	0.00	0.00	0.00
24	0.01	0.00	0.00	0.00	0.00
25	0.01	0.00	0.00	0.00	0.00
26	0.01	0.00	0.00	0.00	0.00
27	0.01	0.00	0.00	0.00	0.00
28	0.01	0.00	0.00	0.00	0.00
29	0.01	0.00	0.00	0.00	0.00
30	0.01	0.00	0.00	0.00	0.00
31	0.01	0.00	0.00	0.00	0.00
32	0.01	0.00	0.00	0.00	0.00
33	0.01	0.00	0.00	0.00	0.00
34	0.01	0.00	0.00	0.00	0.00
35	0.01	0.00	0.00	0.00	0.00
36	0.01	0.00	0.00	0.00	0.00
37	0.01	0.00	0.00	0.00	0.00
38	0.50	0.25	0.17	0.47	0.33
39	0.01	0.00	0.00	0.00	0.00
40	0.01	0.00	0.00	0.00	0.00
41	0.01	0.00	0.00	0.00	0.00
42	0.01	0.00	0.00	0.00	0.00
43	0.01	0.00	0.00	0.00	0.00
44	0.01	0.00	0.01	0.02	0.02
45	0.01	0.00	0.00	0.00	0.00
46	0.01	0.00	0.00	0.00	0.00
47	0.01	0.00	0.00	0.00	0.00
48	0.01	0.00	0.00	0.00	0.00

49	0.01	0.00	0.00	0.00	0.00
50	0.01	0.25	0.48	0.30	0.49
51	0.01	0.00	0.00	0.00	0.00
52	0.01	0.00	0.00	0.00	0.00
53	0.01	0.00	0.00	0.00	0.00
54	0.01	0.00	0.00	0.00	0.00

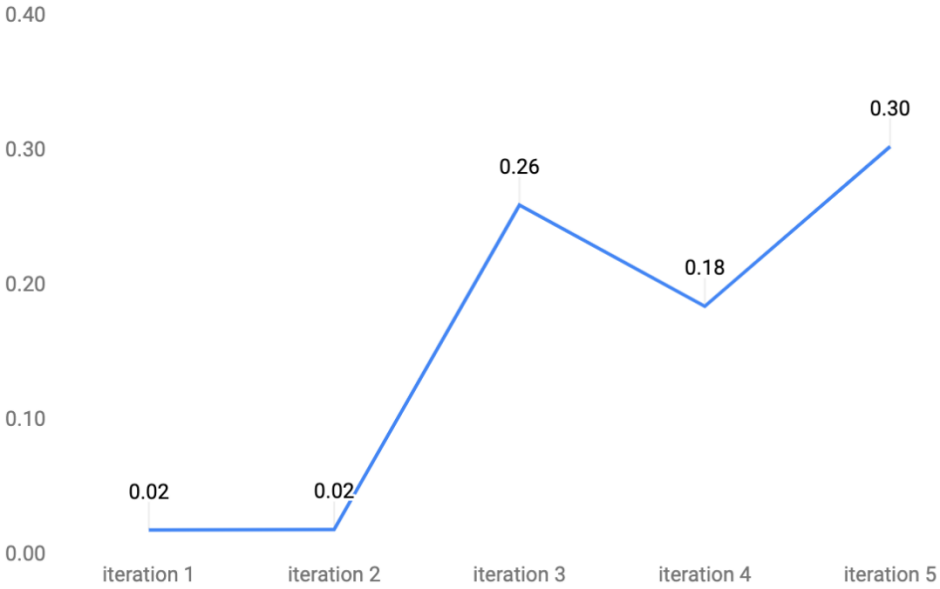

Figure 2: Weighted Errors

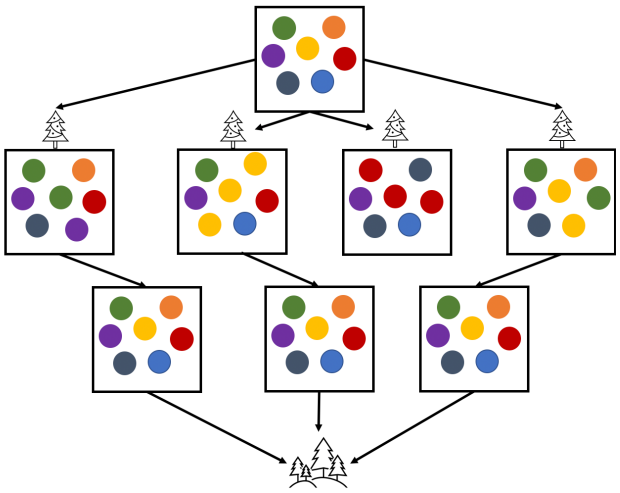

Figure 3: The Boosting-Based pruning Pseudo Code

8. Strengths

1." Early stopping in the aggregation process allows to select subensembles that outperform bagging and retain bagging's resilience to noise"

2. This method reduces the storage needs.

3. This method speeds up the classification process.

9. Drawbacks

1. This is greedy method, so it may lead to suboptimal solutions of the pruning problem.
2. In rare case where at first iteration there is no classifier has a weighted training error below 50%, the algorithm will be in infinite loop.
3. Compared to XGBoost, BB algorithm is slower. However, compared to other ensemble models, it may be faster in the classification process due to the pruning operation

10. Experimental Results

Part C

In this section, we examine the performance of BB and XGBoost on 150 classification datasets. We used 10-Fold Cross-Validation for calculation of the external metric which includes accuracy, TPR, FPR, precision, AUC-ROC, AUC precision-recall, training time, and inference time.We used micro-average to know the performance overall across the data. We used Bayesian Optimization for hyperparameters tuning with 3-Fold Cross-Validation for choosing the best parameters. The best parameters were chosen from Table 4 according to the best accuracy score. We chose those hyperparameters for XGBoost to control overfitting (max_depth,min_child_weight,gamma,learning_rate) and to handle different data sizes(n_estimators). For BB we chose the parameters T (number of classifiers) and U (percentage of classifiers to select) because they are the main parameters in the algorithm. The results of this section are in the attached files (bb_all_final.csv and all_xgb.csv)

Table 4: Hyper parameters

Algorithm	**Hyperparameter name**	**Hyperparameter values**
XGBClassifier	Max_depth	Hyperport.choice('max_depth',range(5,7,1))
XGBClassifier	Learning_rate	Hyperport.quniform('learning_rate,range(0.01,0.5,0.01))
XGBClassifier	N_estimators	Hyperport.choice('max_depth',range(20,205,5))
XGBClassifier	Gamma	Hyperport.quniform('learning_rate,range(0,0.5,0.01))
XGBClassifier	Min_child_weight	Hyperport.quniform('learning_rate,range(1,10,1))
BB	T	Hyperport.choice('max_depth',range(10,150,20))

BB	U	Hyperport.choice('max_depth',range(10,100,20))

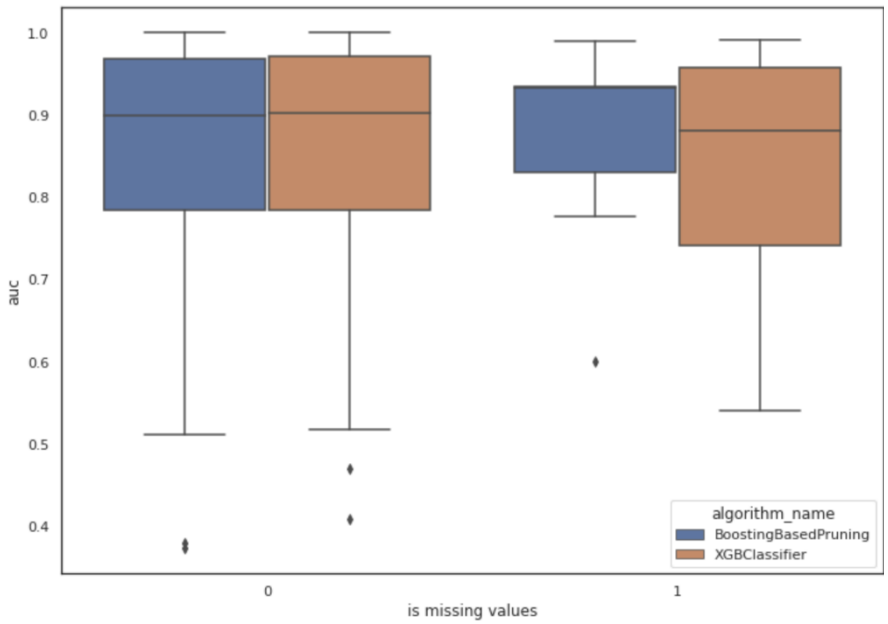

Figure 4: Missing value versus AUC

Table 5: Summary of training and inference time.

	Training time		**Inference time**	
	mean	std	Mean	std
BB	30.6	279.22	10.01	39.28
XGBClassifier	0.42	1.16	0.28	0.53

Part D

A Wilcoxon test was conducted to evaluate whether there was a difference in AUC performance between XGBoost and BB models. Results of that analysis indicated that median XGBoost ranks were statistically higher than the median BB ranks $Z= 7136$, $p<0.001$.

Part E

In this section, we build a meta-learning model which aims to predict the best algorithm (XGBoost or BB) according to Meta-Features features. The target was the winner between the algorithm according to the mean AUC from part C. The evaluation performance of the model was done by Leave-One-Out cross validation. The performance of the model presented in Table 6. weight, gain and cover are in the attached files.

Table 6: Meta-Learning model performance.

accuracy	tpr	fpr	precision	auc	Pr curve	Training time	Inference time
0.6	0.6	0.4	0.6	0.6	0.43	0.14	5.86

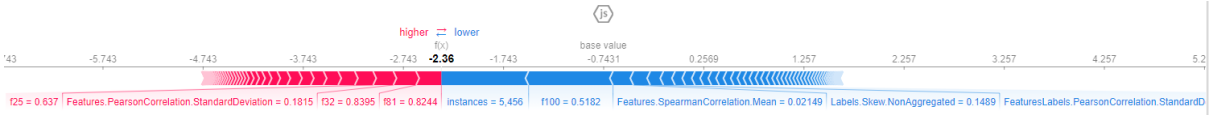

Figure 5: Individual force plot

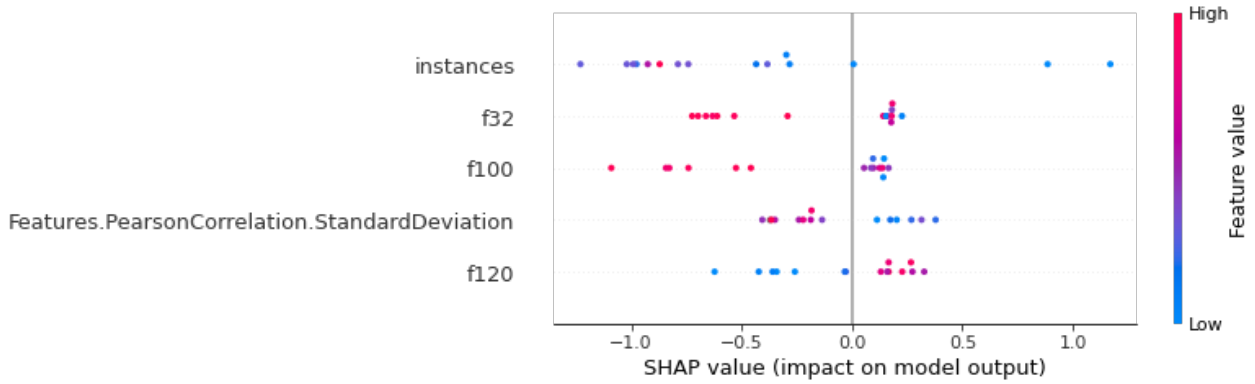

Figure 6: Summary plot

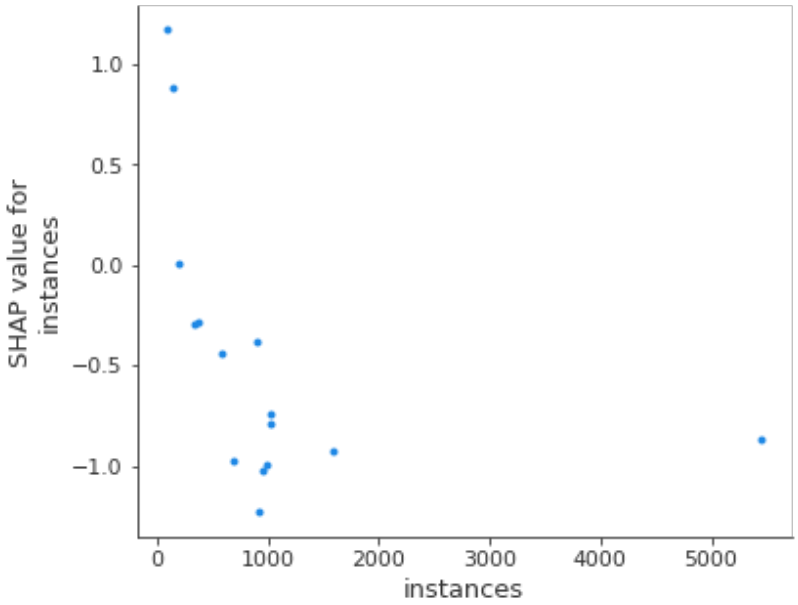

Figure 7: SHAP values for instances

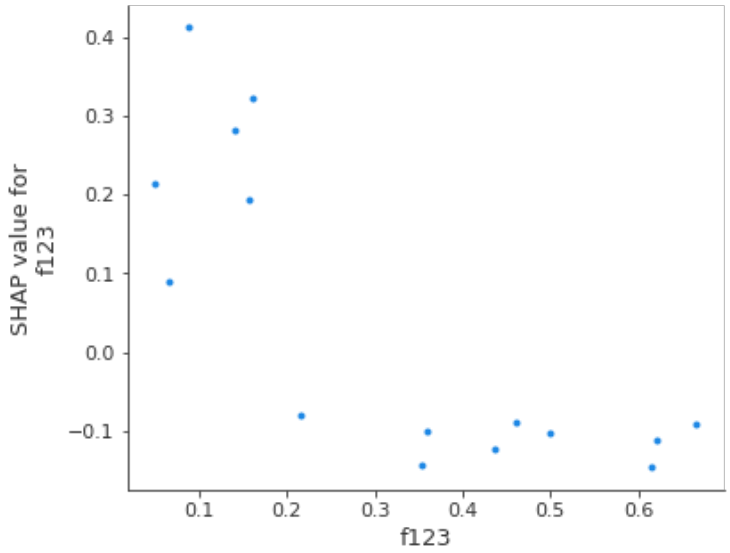

Figure 8: SHAP values for f123

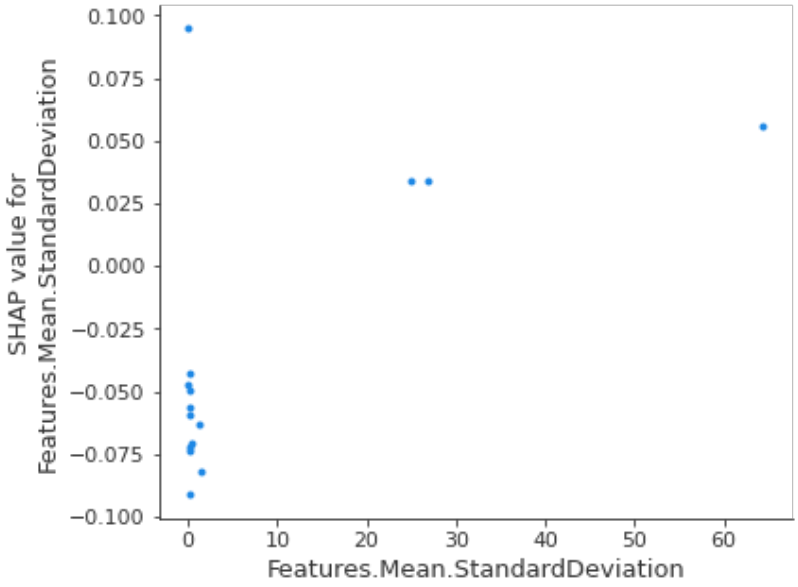

Figure 9: SHAP values for Features.Mean.StandardDeviation

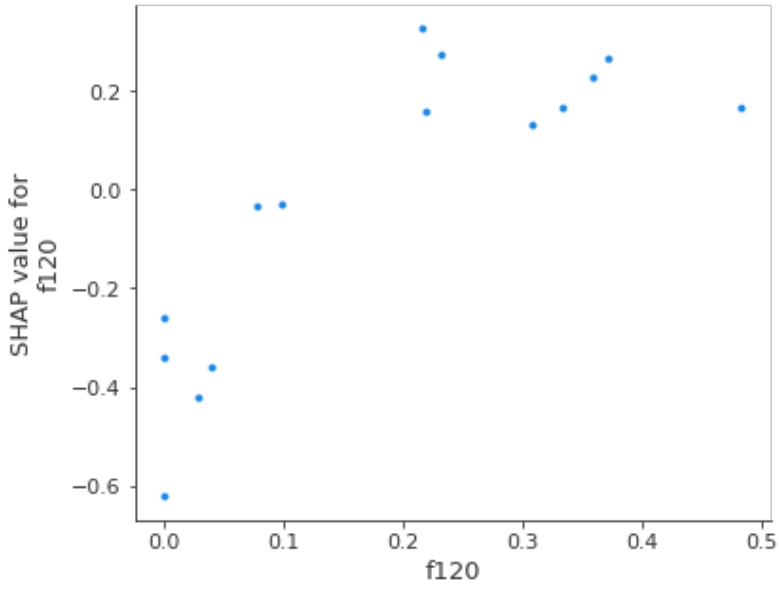

Figure 10: SHAP values for f120

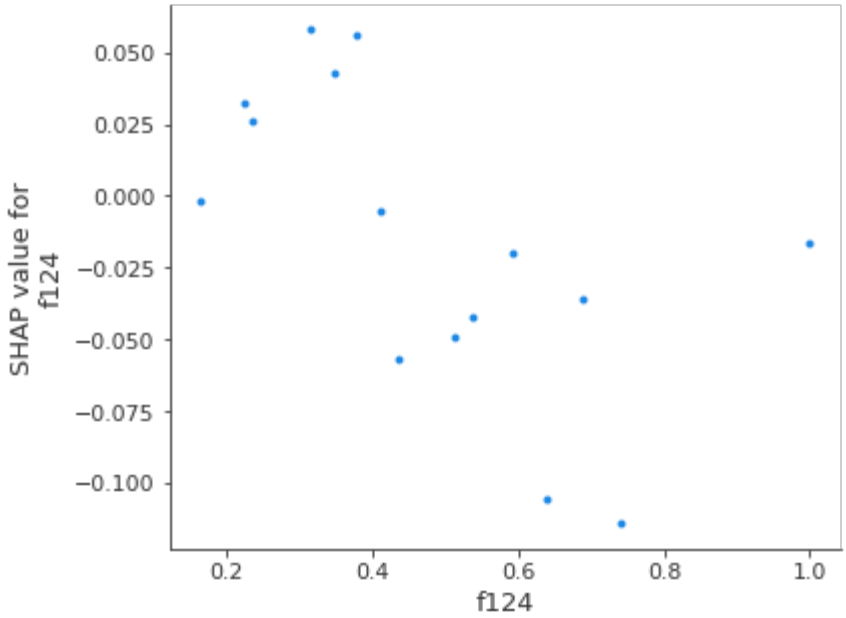

Figure 11: SHAP values for f124

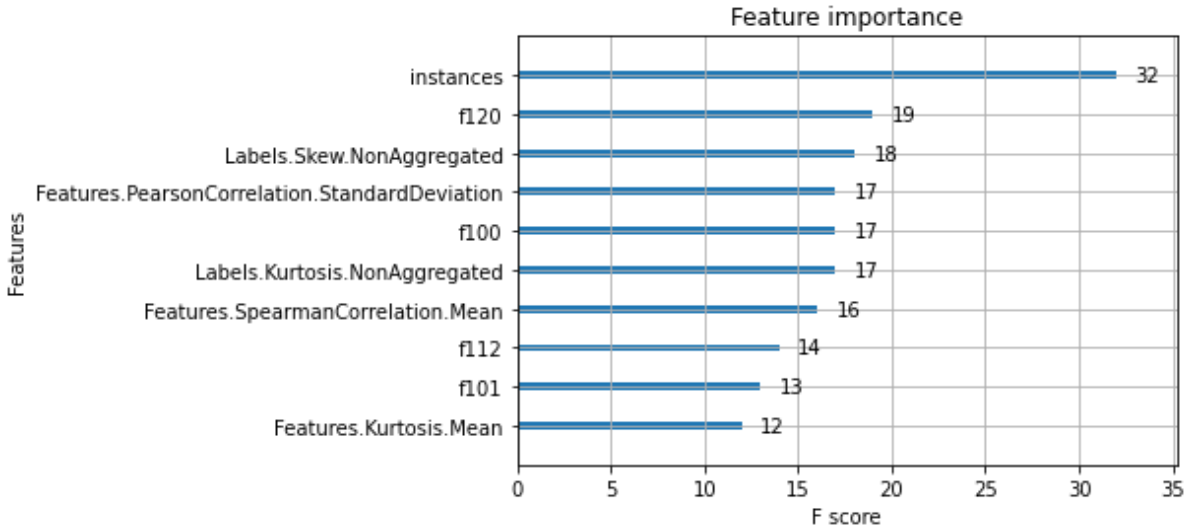

Figure 12: Feature importance

11. Conclusions

The experiments demonstrate that while BB is more time consuming than XGBoost (Table 5), BB is more robust to noisy data (figure 6). For the investigated datasets when there are no missing values, the performances are similar in both algorithms. On the other hand, when there are missing values, BB handles better in comparison to XGBoost, with higher mean AUC values and smaller AUC variance.

In addition, according to the results of the statistical test XGBoost is statistically better in terminology of AUC performance. However, the meta-learning model does not adequately learn which model is better, although it receives better accuracy than random guesses. So in practice, the observations indicated that selecting a subset of classifiers from the original ensemble may lead to an improvement of the classification performance.

By analysis of shap results, we observed that the biggest impact comes from 'instances'. For individual force plot (Figure 5) we predicted -2.36 while the base_value is -0.74.The most egregious features by XGboost feature importance are *FeaturesLabels.PearsonCorrelation.StandardDeviation* with maximum gain, and cover, and *Instances* with maximum weight and F score (Figure 12). Therefore it can be concluded that there is a difference in performance between the models depending on the data size and the feature correlation variance.

12. Citations

Paper	**Citation**
Ulaş, A., Semerci, M., Yıldız, O. T., & Alpaydın, E. (2009). Incremental construction of classifier and discriminant ensembles. *Information Sciences*, *179*(9), 1298-1318.	"Their method outperforms bagging and is comparable to AdaBoost, though when the noise level is high, their method outperforms AdaBoost, which makes it a better alternative ensemble method when the noise level is not known"
Fumera, G., Roli, F., & Serrau, A. (2008). A theoretical analysis of bagging as a linear combination of classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(7), 1293-1299.	"We point out however that selection methods proposed so far are not based on theoretical results, and in particular do not allow to predict the attainable performance improvement over standard bagging"
Dai, Q. (2013). A competitive ensemble pruning approach based on cross-validation technique. *Knowledge-Based Systems*, *37*, 394-414.	"Greedy algorithms, however, possess high speed, since they only consider a very small subspace among all the possible combinations. But this characteristic may result in suboptimal solutions of the ensemble pruning problem"
Hernández-Lobato, D., Martínez-Muñoz, G., & Suárez, A. (2011). Empirical analysis and evaluation of approximate techniques for pruning regression bagging ensembles. *Neurocomputing*, *74*(12-13), 2250-2264.	"The task of selecting from a pool of predictors a subset whose performance is optimal is a difficult problem. On the one hand, it is computationally expensive."
Onan, A., Korukoğlu, S., & Bulut, H. (2017). A hybrid ensemble pruning approach based on consensus clustering	"The experimental results indicated that the pruned ensemble can improve the classification speed, reduce

and multi-objective evolutionary algorithm for sentiment classification. *Information Processing & Management*, *53*(4), 814-833.	the memory requirements and enhance the predictive performance."
Kotsiantis, S. B. (2014). Bagging and boosting variants for handling classifications problems: a survey. *The Knowledge Engineering Review*, *29*(1), 78.	"However, in problems where boosting is superior to bagging, these improvements were not sufficient to reach the accuracy of the corresponding boosting ensembles"
Zhang, H., & Cao, L. (2014). A spectral clustering based ensemble pruning approach. *Neurocomputing*, *139*, 289-297	"Studies show that it is possible to achieve a small yet strong ensemble"
Dai, Q., Zhang, T., & Liu, N. (2015). A new reverse reduce-error ensemble pruning algorithm. *Applied Soft Computing*, *28*, 237-249.	"Although greedy algorithms possess relatively high efficiency, as they only consider a very small subspace within the entire solution space, they might receive suboptimal solutions of the ensemble pruning problem"
Adnan, M. N., & Islam, M. Z. (2016). Optimizing the number of trees in a decision forest to discover a subforest with high ensemble accuracy using a genetic algorithm. *Knowledge-Based Systems*, *110*, 86-97.	"In literature, subforests are generated particularly from Bagging ensembles citing it's robustness"
Zhou, Hongfang, Xuehan Zhao, and Xiao Wang. "An effective ensemble pruning algorithm based on frequent patterns." *Knowledge-Based Systems* 56 (2014): 79-85.	"Moreover, Greedy algorithm is also one of the extensively investigated topics related to ensemble pruning. It processes quickly only considering a small subspace among all the possible combinations. However, this would cause a strong possibility of suboptimal solution of the ensemble pruning"