Computational Learning

Adva Segal 200849511 Yam Sharon 208609198
GitHub - https://github.com/yamsharon102/Computation-Learning---Project

ALGORITHM NAME

Dynamic boosted ensemble learning method based on random forest (DBRF)

REFERENCE

Ren, X., Long, C., Zhang, L., Wei, Y., Du, D., Liang, J., ... & Li, W. (2018). A dynamic boosted ensemble learning method based on random forest. *arXiv preprint arXiv:1804.07270*.

THE MOTIVATION FOR THE ALGORITHM

Many machine learning algorithms use the notion of hard examples of mining (e.g., AdaBoost). This paper is motivated to use this notion into random forests (RF). Rather than assigning more weight to hard examples, they remove these data from the training data. In this way, they make the model focus on hard examples and learn decision boundaries better without making it more complicated.

SHORT DESCRIPTION:

"It is a novel ensemble algorithm where such notion of hard example mining as in Boosting is incorporated into RF. Specifically, we propose to score the quality of each leaf node of every decision tree in the random forest and then vote to determine hard examples. By iteratively training and then removing easy examples from training data to train again, we evolve the model to focus on hard examples dynamically so as to learn decision boundaries better. After training, the test data can be cascaded through these learned random forests of each iteration in sequence to generate predictions, thus achieving more depth with RF."

This algorithm uses several RF models and takes advantage of the strength of each RF to predict its easy examples.

PSEUDO-CODE

Dynamic Boosted Random Forest Algorithm

Input : *Training set* : *Test set* : D^t , *Iterations n*

Output : Prediction of test set : O

1. //Training Procedure
2. $D \leftarrow D^d$
3. **for** $i = 1 \rightarrow n$ **do**
4. *Train RF as* F_i *on dataset D*
5. Get scores of leaf nodes i by HEM
6. Split D into easy data D_e and hard data D_h according to F_i and π_i
7. $D \leftarrow D_h$
8. Add F_i to F - list
9. Add π_i to π - List
10. End for
11.
12. // Test Procedure
13. $D \leftarrow D^t$
14. $O \leftarrow \phi$
15. for $i = 1 \rightarrow n$ do
16. $F_i \leftarrow$ F - list at i
17. $\pi_i \leftarrow \pi$ - list at i
18. Split D into easy data D_e and hard data D_h according to F_i and π_i
19. Predict D_e as O' by F_i
20. $D \leftarrow D_h$
21. $O = O \cup O'$
22. end for
23. Predict D_h as O' by F_n
24. $O = O \cup O'$
25. return O

Hard Example Mining (HEM) Algorithm

$$D^e{}_F = \bigcap_{T \in F} D^e{}_T \, , D^h{}_F = D \setminus D^e{}_F$$

$$D^e{}_T = \{x \mid x \in \bigcap_{Leaf \in T, T \in F} (score(Leaf) > \sigma)\}$$

Where *score*() is a leaf node evaluation metric, and σ is the threshold, which is implemented as the average score of lead nodes in all decision trees in our model.

$$score(L) = 2 * \frac{score_{supp}(L) * score_{conf}(L)}{score_{supp}(L) + score_{conf}(L)}$$

$$score_{supp}(L) = \frac{|C|}{|X|}$$

$$score_{conf}(L) = \frac{\{y = c, x \epsilon C\}}{|C|}$$

Where C is a candidate set.

ALGORITHM EXPLANATION

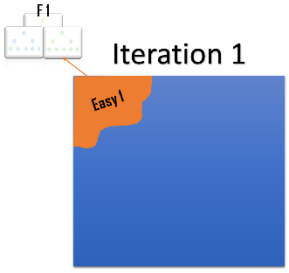
- **Hard Example Mining (HEM)** – this algorithm's goal is to score every leaf of the random forest model to determine what examples are easy and what examples are hard. To do so, the paper's authors proposed a formula that expresses the confidence of the leaf in the given examples. With this score, we can examine if some example belongs only to good leaves, and then we will know that this example is easy for the specific RF.
- **Dynamic Boosted Random Forest Algorithm (DBRF)** –
    - **Training Phase** – In every iteration i, we train an RF F_i (line 4) on the remaining examples (that was yet to labeled as easy). Using F_i, we give scores for every leaf (line 5) to be able to label every example as easy or hard (line 6). We save the RF model and the leaf scores (π_i) for every iteration (lines 8-9).
    - **Test Phase** – In every iteration i, we take Fi and π_i (line 16-17) and split the remaining examples to easy and hard. Then, we will only predict the easy examples. Eventually, we will predict the remaining example with F_n.

ILLUSTRATION

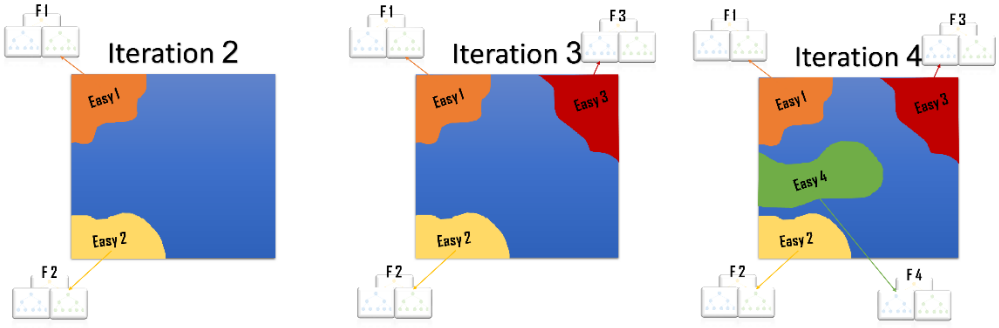
In this section, we will review a general execution of the DBRF algorithm.

1. At first, let us assume we want to fit our algorithm to the dataset – D –

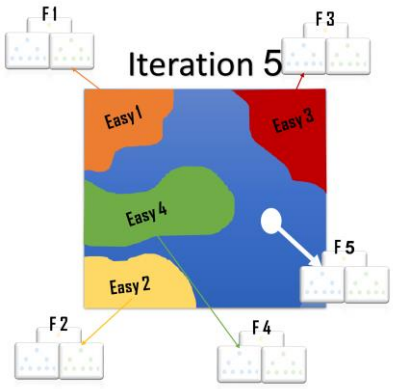
2. At the first iteration, we train a Random Forest (F1) model on the whole dataset. With the HEM algorithm, we will find the easy examples (Easy 1) for this specific RF and will save the RF and the leaf scores which we got from the HEM.



3. Then, we will make n iterations, and for each iteration, we will use only the rest of the examples from the dataset (Those that we did not find as easy) and will save the RF models (F2, F3, F4) and leaf scores.



4. Eventually, we will fit an RF (F5) model for the rest of the examples in the dataset –



5. To predict new examples, we will make n iterations. For each iteration i, we will use F_i to determine the easy examples of the test set, and we will predict these examples.
6. Eventually, in iteration n, we will predict the rest of the examples with F_n.

Execution Example –

Initial dataset-

	f1	f2	f3	f4	f5	f6	my_new_class	
0	-1.77236	-0.562162	0.841625	-1.408310	-0.979364	-0.841625	0	
1	-1.55248	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
2	-1.55248	-0.562162	0.841625	-1.408310	-0.979364	-0.841625	0	
3	-1.49751	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
4	-1.49751	-0.562162	0.841625	-1.408310	-0.979364	-0.841625	0	
...	...	...	...	...	...	...	...	...
115	1.47094	-0.562162	0.841625	0.704154	-0.979364	1.178280	0	
116	1.52591	-0.562162	-1.178280	-1.408310	-0.979364	-0.841625	0	
117	1.52591	1.764020	0.841625	-1.408310	1.012560	-0.841625	0	

	f1	f2	f3	f4	f5	f6	my_new_class	
118	1.52591	-0.562162	0.841625	0.704154	-0.979364	1.178280	0	
119	1.52591	-0.562162	0.841625	0.704154	-0.979364	1.178280	0	

120 rows × 7 columns

After 1st iteration –

	f1	f2	f3	f4	f5	f6	my_new_class	
1	-1.55248	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
3	-1.49751	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
6	-1.38757	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
8	-1.33259	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
9	-1.16768	-0.562162	-1.178280	0.704154	1.012560	1.178280	1	
...	...	...	...	...	...	...	...	...
115	1.47094	-0.562162	0.841625	0.704154	-0.979364	1.178280	0	
116	1.52591	-0.562162	-1.178280	-1.408310	-0.979364	-0.841625	0	
117	1.52591	1.764020	0.841625	-1.408310	1.012560	-0.841625	0	
118	1.52591	-0.562162	0.841625	0.704154	-0.979364	1.178280	0	
119	1.52591	-0.562162	0.841625	0.704154	-0.979364	1.178280	0	

96 rows × 7 columns

After the second iteration –

	f1	f2	f3	f4	f5	f6	my_new_class
70	0.701342	1.764020	0.841625	0.704154	1.012560	1.178280	1
71	0.701342	1.764020	0.841625	0.704154	1.012560	1.178280	1
72	0.701342	1.764020	0.841625	0.704154	1.012560	-0.841625	1
75	0.701342	1.764020	0.841625	-1.408310	1.012560	-0.841625	0
76	0.701342	1.764020	0.841625	-1.408310	1.012560	-0.841625	0
78	0.756313	1.764020	0.841625	0.704154	1.012560	-0.841625	1
79	0.811284	1.764020	0.841625	0.704154	1.012560	1.178280	1
...	...	...	...	...	...	...	...
114	1.415970	1.764020	0.841625	0.704154	1.012560	-0.841625	1
115	1.470940	-0.562162	0.841625	0.704154	-0.979364	1.178280	0
116	1.525910	-0.562162	-1.178280	-1.408310	-0.979364	-0.841625	0
117	1.525910	1.764020	0.841625	-1.408310	1.012560	-0.841625	0
118	1.525910	-0.562162	0.841625	0.704154	-0.979364	1.178280	0
119	1.525910	-0.562162	0.841625	0.704154	-0.979364	1.178280	0

40 rows × 7 columns

(We can see the size of the remaining of the dataset is decreasing in every iteration)

STRENGTHS

- "We believe that DBRF is a very practical approach and is particularly useful in learning from imbalanced data."
- "novel ensemble algorithm where such notion of hard example mining as in Boosting is incorporated into RF."
- The algorithm uses the strength of each Random Forest instance.
- The algorithm correlates between the (probably) best path in a random forest and an instance.

DRAWBACKS

- "The training time of DBRF is quite competitive" regards to other boosting algorithms – Much greater than standard classifications methods – It uses and fits many RF models.
- Most times, do not find enough easy examples.
- The algorithm does not show significant results on normal (balanced) datasets.

RESULTS

Attached in the file – Best results.docx

STATISTICAL SIGNIFICANCE OF THE RESULTS

Since we chose to Implement an algorithm, we compare between only two algorithms. So Instead of Friedman Test and Post-Hoc we used Wilcoxon Signed-Rank Test.

Quote from the site: https://machinelearningmastery.com/nonparametric-statistical-significance-tests-in-python/

Wilcoxon Signed-Rank Test

…Examples of paired samples in machine learning might be the same algorithm evaluated on different datasets or different algorithms evaluated on exactly the same training and test data.

The samples are not independent; therefore, the Mann-Whitney U test cannot be used. Instead, the Wilcoxon signed-rank test is used, also called the Wilcoxon T test, named for Frank Wilcoxon. It is the equivalent of the paired Student T-test, but for ranked data instead of real valued data with a Gaussian distribution.

The Wilcoxon signed ranks test is a nonparametric statistical procedure for comparing two samples that are paired, or related. The parametric equivalent to the Wilcoxon signed ranks test goes by names such as the Student's t-test, t-test for matched pairs, t-test for paired samples, or t-test for dependent samples.

— Pages 38-39, Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach, 2009.
The default assumption for the test, the null hypothesis, is that the two samples have the same distribution.

- Fail to Reject H0: Sample distributions are equal.
- Reject H0: Sample distributions are not equal.

For the test to be effective, it requires at least 20 observations in each data sample.

The Wilcoxon signed-rank test can be implemented in Python using the wilcoxon() SciPy function. The function takes the two samples as arguments and returns the calculated statistic and p-value.
The complete example is below, demonstrating the calculation of the Wilcoxon signed-rank test on the test problem. The two samples are technically not paired, but we can pretend they are for the sake of demonstrating the calculation of this significance test.

The p-value is interpreted strongly suggesting that the samples are drawn from different distributions.

```
Statistics=543.000, p=0.000
Different distribution (reject H0)
```

BUILDING A LEARNING-META MODEL

We used XGBoost algorithm for building Learning-Meta model.

Building Training data

First, we slice the results from phase C to DBRF algorithm and Extra Trees. Then we calculate the max AUC result for each 10-Fold cross validation for each dataset (max instead of average since the Extra Trees algorithm I much better than DBRF and we want to give chance for DBRF also). Who get the highest value is the winner and its label is 1. In this stage we have a data

frame with algorithm name, data sat name and labels. In the next stage we joined this data frame with the given Meta-Features and drop the data set name column.

Leave-one-out protocol results

Dataset Name	Algorithm Name	Accuracy	TPR	FPR	Precision	AUC	PR-Curve	Training Time	Inference Time
abalone	XGBoost	1	1	0	1	1	1	1.109	54.688
acute-inflammation	XGBoost	1	1	0	1	1	1	1.250	0.000
acute-nephritis	XGBoost	1	1	0	1	1	1	1.125	0.000
analcatdata_asbestos	XGBoost	1	1	0	1	1	1	2.453	62.500
analcatdata_boxing1	XGBoost	1	1	0	1	1	1	2.375	101.563
analcatdata_broadwaymult	XGBoost	1	1	0	1	1	1	2.500	0.000
analcatdata_germangss	XGBoost	0	0	1	0	0	0.5	2.547	0.000
analcatdata_lawsuit	XGBoost	1	1	0	1	1	1	2.344	62.500
annealing	XGBoost	1	1	0	1	1	1	2.375	62.500
ar4	XGBoost	1	1	0	1	1	1	2.359	46.875
arrhythmia	XGBoost	1	1	0	1	1	1	2.313	62.500
audiology-std	XGBoost	0	0	1	0	0	0.5	2.297	62.500
autos	XGBoost	1	1	0	1	1	1	2.234	15.625
balance-scale	XGBoost	0	0	1	0	0	0.5	2.500	62.500
bank	XGBoost	1	1	0	1	1	1	2.484	62.500
baseball	XGBoost	1	1	0	1	1	1	2.484	39.063
blood	XGBoost	1	1	0	1	1	1	2.531	0.000
bodyfat	XGBoost	1	1	0	1	1	1	2.453	39.063
braziltourism	XGBoost	1	1	0	1	1	1	2.547	0.000
breast-cancer	XGBoost	1	1	0	1	1	1	2.500	7.813
breast-cancer-wisc	XGBoost	1	1	0	1	1	1	2.344	7.813
breast-cancer-wisc-diag	XGBoost	1	1	0	1	1	1	2.547	0.000
breast-cancer-wisc-prog	XGBoost	1	1	0	1	1	1	2.328	46.875
breast-tissue	XGBoost	1	1	0	1	1	1	2.328	62.500
car	XGBoost	1	1	0	1	1	1	2.422	62.500
cardiotocography-10clases	XGBoost	1	1	0	1	1	1	2.469	0.000
cardiotocography-3clases	XGBoost	1	1	0	1	1	1	2.484	31.250
chatfield_4	XGBoost	1	1	0	1	1	1	2.219	0.000
chess-krvkp	XGBoost	1	1	0	1	1	1	2.422	62.500
chscase_vine1	XGBoost	1	1	0	1	1	1	2.359	62.500
cloud	XGBoost	1	1	0	1	1	1	2.469	0.000
congressional-voting	XGBoost	1	1	0	1	1	1	2.344	54.688
conn-bench-sonar-mines-rocks	XGBoost	1	1	0	1	1	1	2.453	23.438
conn-bench-vowel-deterding	XGBoost	1	1	0	1	1	1	2.281	0.000
contrac	XGBoost	0	0	1	0	0	0.5	2.531	62.500
credit-approval	XGBoost	1	1	0	1	1	1	2.500	0.000
cylinder-bands	XGBoost	1	1	0	1	1	1	2.375	62.500
dermatology	XGBoost	1	1	0	1	1	1	2.375	62.500
diabetes	XGBoost	1	1	0	1	1	1	2.438	23.438
diggle_table_a2	XGBoost	1	1	0	1	1	1	2.297	54.688
disclosure_z	XGBoost	1	1	0	1	1	1	2.531	0.000
echocardiogram	XGBoost	1	1	0	1	1	1	2.422	62.500
ecoli	XGBoost	1	1	0	1	1	1	2.406	23.438
elusage	XGBoost	1	1	0	1	1	1	2.422	62.500

energy-y1	XGBoost	1	1	0	1	1	1	2.250	7.813
energy-y2	XGBoost	1	1	0	1	1	1	2.438	62.500
fertility	XGBoost	1	1	0	1	1	1	2.313	39.063
flags	XGBoost	1	1	0	1	1	1	2.313	62.500
fri_c0_250_5	XGBoost	1	1	0	1	1	1	2.422	62.500
glass	XGBoost	1	1	0	1	1	1	2.422	7.813
haberman-survival	XGBoost	1	1	0	1	1	1	2.469	0.000
hayes-roth	XGBoost	0	0	1	0	0	0.5	2.469	62.500
heart-cleveland	XGBoost	1	1	0	1	1	1	2.328	62.500
heart-hungarian	XGBoost	1	1	0	1	1	1	2.375	62.500
heart-switzerland	XGBoost	1	1	0	1	1	1	2.484	0.000
heart-va	XGBoost	1	1	0	1	1	1	2.422	62.500
hepatitis	XGBoost	1	1	0	1	1	1	2.406	62.500
hill-valley	XGBoost	1	1	0	1	1	1	2.359	0.000
horse-colic	XGBoost	1	1	0	1	1	1	2.406	0.000
ilpd-indian-liver	XGBoost	1	1	0	1	1	1	2.609	0.000
image-segmentation	XGBoost	1	1	0	1	1	1	2.453	62.500
ionosphere	XGBoost	1	1	0	1	1	1	2.391	7.813
iris	XGBoost	1	1	0	1	1	1	2.516	62.500
kc3	XGBoost	1	1	0	1	1	1	2.406	54.688
kidney	XGBoost	1	1	0	1	1	1	2.313	62.500
labor	XGBoost	1	1	0	1	1	1	2.391	0.000
led-display	XGBoost	0	0	1	0	0	0.5	2.266	54.688
lenses	XGBoost	1	1	0	1	1	1	2.344	62.500
libras	XGBoost	1	1	0	1	1	1	2.375	62.500
lowbwt	XGBoost	1	1	0	1	1	1	2.359	62.500
low-res-spect	XGBoost	1	1	0	1	1	1	2.375	62.500
lung-cancer	XGBoost	1	1	0	1	1	1	2.219	62.500
lupus	XGBoost	1	1	0	1	1	1	2.500	0.000
lymphography	XGBoost	1	1	0	1	1	1	2.438	7.813
mammographic	XGBoost	1	1	0	1	1	1	2.266	62.500
meta	XGBoost	1	1	0	1	1	1	2.375	85.938
mfeat-karhunen	XGBoost	0	0	1	0	0	0.5	2.391	62.500
mfeat-morphological	XGBoost	1	1	0	1	1	1	2.359	62.500
molec-biol-promoter	XGBoost	1	1	0	1	1	1	2.281	62.500
molec-biol-splice	XGBoost	1	1	0	1	1	1	2.406	7.813
monks-1	XGBoost	1	1	0	1	1	1	2.391	7.813
monks-2	XGBoost	1	1	0	1	1	1	2.188	46.875
monks-3	XGBoost	1	1	0	1	1	1	2.266	62.500
mushroom	XGBoost	1	1	0	1	1	1	2.266	62.500
musk-1	XGBoost	1	1	0	1	1	1	2.375	54.688
newton_hema	XGBoost	1	1	0	1	1	1	2.297	46.875

no2	XGBoost	1	1	0	1	1	1	2.344	0.000
oocytes_merluccius_nucleus_4d	XGBoost	1	1	0	1	1	1	2.375	0.000
oocytes_merluccius_states_2f	XGBoost	1	1	0	1	1	1	2.469	62.500
oocytes_trisopterus_nucleus_2f	XGBoost	1	1	0	1	1	1	2.453	7.813
oocytes_trisopterus_states_5b	XGBoost	1	1	0	1	1	1	2.422	7.813
ozone	XGBoost	1	1	0	1	1	1	2.266	62.500
parkinsons	XGBoost	1	1	0	1	1	1	2.453	62.500
pima	XGBoost	1	1	0	1	1	1	2.297	0.000
pittsburg-bridges-MATERIAL	XGBoost	1	1	0	1	1	1	2.484	0.000
pittsburg-bridges-REL-L	XGBoost	0	0	1	0	0	0.5	2.563	62.500
pittsburg-bridges-SPAN	XGBoost	0	0	1	0	0	0.5	2.453	62.500
pittsburg-bridges-T-OR-D	XGBoost	1	1	0	1	1	1	2.375	62.500
pittsburg-bridges-TYPE	XGBoost	0	0	1	0	0	0.5	2.344	62.500
planning	XGBoost	1	1	0	1	1	1	2.250	62.500
plant-margin	XGBoost	0	0	1	0	0	0.5	2.531	7.813
plant-shape	XGBoost	1	1	0	1	1	1	2.484	39.063
plant-texture	XGBoost	1	1	0	1	1	1	2.375	31.250
plasma_retinol	XGBoost	1	1	0	1	1	1	2.500	0.000
pm10	XGBoost	1	1	0	1	1	1	2.375	39.063
post-operative	XGBoost	1	1	0	1	1	1	2.266	39.063
prnn_synth	XGBoost	1	1	0	1	1	1	2.266	46.875
rabe_131	XGBoost	1	1	0	1	1	1	2.469	62.500
rmftsa_sleepdata	XGBoost	1	1	0	1	1	1	2.375	46.875
schizo	XGBoost	1	1	0	1	1	1	2.484	7.813
schlvote	XGBoost	1	1	0	1	1	1	2.313	46.875
seeds	XGBoost	1	1	0	1	1	1	2.313	7.813
semeion	XGBoost	1	1	0	1	1	1	2.406	31.250
sleuth_case2002	XGBoost	1	1	0	1	1	1	2.344	54.688
socmob	XGBoost	1	1	0	1	1	1	2.422	62.500
solar-flare	XGBoost	1	1	0	1	1	1	2.359	0.000
soybean	XGBoost	1	1	0	1	1	1	2.484	15.625
spambase	XGBoost	1	1	0	1	1	1	2.297	7.813
spect	XGBoost	1	1	0	1	1	1	2.375	0.000
spectf	XGBoost	0	0	1	0	0	0.5	2.391	0.000
squash-stored	XGBoost	1	1	0	1	1	1	2.438	0.000
squash-unstored	XGBoost	1	1	0	1	1	1	2.422	62.500
statlog-australian-credit	XGBoost	1	1	0	1	1	1	2.375	62.500
statlog-german-credit	XGBoost	1	1	0	1	1	1	2.438	0.000
statlog-heart	XGBoost	1	1	0	1	1	1	2.406	62.500
statlog-image	XGBoost	1	1	0	1	1	1	2.203	62.500
statlog-vehicle	XGBoost	1	1	0	1	1	1	2.391	62.500
steel-plates	XGBoost	1	1	0	1	1	1	2.422	0.000

synthetic-control	XGBoost	1	1	0	1	1	1	2.344	62.500
tae	XGBoost	1	1	0	1	1	1	2.328	0.000
teaching	XGBoost	1	1	0	1	1	1	2.375	62.500
teachingAssistant	XGBoost	1	1	0	1	1	1	2.375	23.438
tic-tac-toe	XGBoost	1	1	0	1	1	1	2.203	62.500
titanic	XGBoost	1	1	0	1	1	1	2.422	0.000
transplant	XGBoost	1	1	0	1	1	1	2.469	0.000
triazines	XGBoost	1	1	0	1	1	1	2.359	62.500
vertebral-column-2clases	XGBoost	1	1	0	1	1	1	2.375	62.500
vertebral-column-3clases	XGBoost	1	1	0	1	1	1	2.359	39.063
veteran	XGBoost	1	1	0	1	1	1	2.438	0.000
visualizing_livestock	XGBoost	1	1	0	1	1	1	2.391	62.500
vote	XGBoost	1	1	0	1	1	1	2.406	15.625
wall-following	XGBoost	1	1	0	1	1	1	2.266	62.500
waveform	XGBoost	1	1	0	1	1	1	2.266	62.500
waveform-noise	XGBoost	1	1	0	1	1	1	2.453	62.500
white-clover	XGBoost	1	1	0	1	1	1	2.250	23.438
wine	XGBoost	1	1	0	1	1	1	2.438	39.063
wine_quality_red	XGBoost	1	1	0	1	1	1	2.266	62.500
wine-quality-white	XGBoost	1	1	0	1	1	1	2.375	62.500
yeast	XGBoost	1	1	0	1	1	1	2.406	23.438
zoo	XGBoost	0	0	1	0	0	0.5	2.500	0.000

Importance

Algorithm Name_DBRF feature get very height weight because Extrs Tree algorithm is much more better in most cases so it high correlated to label column.

importance_type	gain	cover	weight
f1	0.000	0.051	0.002
f2	0.000	0.000	0.000
f3	0.000	0.000	0.000
f4	0.000	0.000	0.000
f5	0.018	0.004	0.069
f6	0.000	0.000	0.000
f7	0.000	0.000	0.000
f8	0.000	0.000	0.000
f9	0.000	0.000	0.000
f10	0.000	0.000	0.000
f11	0.000	0.000	0.000
f12	0.000	0.000	0.000
f13	0.000	0.000	0.000

f14	0.000	0.000	0.000
f15	0.000	0.000	0.000
f16	0.000	0.000	0.000
f17	0.000	0.000	0.000
f18	0.043	0.008	0.033
f19	0.000	0.000	0.000
f20	0.000	0.000	0.000
f21	0.018	0.013	0.052
f22	0.000	0.000	0.000
f23	0.000	0.000	0.000
f24	0.000	0.000	0.000
f25	0.000	0.000	0.000
f26	0.000	0.000	0.000
f27	0.000	0.000	0.000
f28	0.000	0.000	0.000
f29	0.000	0.000	0.000
f30	0.000	0.000	0.000
f31	0.000	0.000	0.000
f32	0.000	0.000	0.000
f33	0.000	0.000	0.000
f34	0.000	0.000	0.000
f35	0.000	0.000	0.000
f36	0.002	0.042	0.011
f37	0.000	0.000	0.000
f38	0.000	0.000	0.000
f39	0.000	0.000	0.000
f40	0.000	0.000	0.000
f41	0.000	0.000	0.000
f42	0.000	0.000	0.000
f43	0.000	0.000	0.000
f44	0.000	0.000	0.000
f45	0.000	0.000	0.000
f46	0.000	0.000	0.000
f47	0.000	0.000	0.000
f48	0.000	0.000	0.000
f49	0.019	0.056	0.059
f50	0.000	0.000	0.000
f51	0.000	0.000	0.000
f52	0.000	0.053	0.002
f53	0.000	0.000	0.000
f54	0.000	0.000	0.000
f55	0.000	0.000	0.000

f56	0.000	0.000	0.000
f57	0.000	0.000	0.000
f58	0.000	0.000	0.000
f59	0.000	0.000	0.000
f60	0.000	0.000	0.000
f61	0.016	0.029	0.027
f62	0.000	0.000	0.000
f63	0.000	0.000	0.000
f64	0.000	0.048	0.050
f65	0.003	0.047	0.002
f66	0.000	0.000	0.000
f67	0.000	0.000	0.000
f68	0.000	0.000	0.000
f69	0.000	0.000	0.000
f70	0.000	0.000	0.000
f71	0.000	0.000	0.000
f72	0.000	0.000	0.000
f73	0.015	0.004	0.004
f74	0.000	0.000	0.000
f75	0.000	0.000	0.000
f76	0.000	0.000	0.000
f77	0.000	0.000	0.000
f78	0.000	0.000	0.000
f79	0.000	0.000	0.000
f80	0.013	0.054	0.035
f81	0.000	0.000	0.000
f82	0.000	0.000	0.000
f83	0.000	0.000	0.000
f84	0.011	0.032	0.006
f85	0.000	0.000	0.000
f86	0.000	0.000	0.000
f87	0.007	0.048	0.017
f88	0.000	0.000	0.000
f89	0.000	0.000	0.000
f90	0.000	0.000	0.000
f91	0.000	0.000	0.000
f92	0.000	0.000	0.000
f93	0.000	0.000	0.000
f94	0.000	0.000	0.000
f95	0.000	0.000	0.000
f96	0.000	0.000	0.000
f97	0.004	0.004	0.003

f98	0.000	0.000	0.000
f99	0.000	0.000	0.000
f100	0.000	0.000	0.000
f101	0.000	0.000	0.000
f102	0.000	0.000	0.000
f103	0.000	0.000	0.000
f104	0.000	0.000	0.000
f105	0.000	0.000	0.000
f106	0.000	0.000	0.000
f107	0.002	0.050	0.015
f108	0.000	0.000	0.000
f109	0.000	0.000	0.000
f110	0.000	0.000	0.000
f111	0.004	0.004	0.001
f112	0.002	0.013	0.002
f113	0.000	0.000	0.000
f114	0.000	0.000	0.000
f115	0.000	0.000	0.000
f116	0.016	0.004	0.001
f117	0.000	0.000	0.000
f118	0.000	0.000	0.000
f119	0.000	0.000	0.000
f120	0.000	0.000	0.000
f121	0.000	0.000	0.000
f122	0.000	0.000	0.000
f123	0.012	0.036	0.125
f124	0.000	0.000	0.000
f125	0.000	0.000	0.000
Features.Skew.StandardDeviation	0.000	0.000	0.000
Features.Skew.Mean	0.014	0.055	0.145
Labels.Skew.NonAggregated	0.000	0.000	0.000
Features.PearsonCorrelation.Mean	0.000	0.000	0.000
Features.PearsonCorrelation.StandardDeviation	0.018	0.020	0.061
FeaturesLabels.PearsonCorrelation.Mean	0.013	0.056	0.013
FeaturesLabels.PearsonCorrelation.StandardDeviation	0.000	0.000	0.000
Features.SpearmanCorrelation.Mean	0.005	0.004	0.022
Features.SpearmanCorrelation.StandardDeviation	0.000	0.000	0.000
FeaturesLabels.SpearmanCorrelation.Mean	0.011	0.056	0.018
FeaturesLabels.SpearmanCorrelation.StandardDeviation	0.000	0.000	0.000
Features.Kurtosis.Mean	0.002	0.038	0.006
Features.Kurtosis.StandardDeviation	0.000	0.000	0.000
Labels.Kurtosis.NonAggregated	0.000	0.000	0.000

Features.Mean.Mean	0.000	0.000	0.000
Features.Mean.StandardDeviation	0.004	0.004	0.006
Labels.Mean.NonAggregated	0.018	0.055	0.125
Unnamed: 143	0.000	0.000	0.000
instances	0.000	0.000	0.000
dimensionality	0.000	0.000	0.000
instances_with_missing_val	0.000	0.000	0.000
ratio_of_discrete_features	0.000	0.000	0.000
ratio_of_numeric_features	0.000	0.000	0.000
nonzero_vals_cnt	0.000	0.000	0.000
missing values	0.000	0.000	0.000
attributes total	0.000	0.000	0.000
attributes categorical	0.000	0.000	0.000
attributes numerical	0.000	0.000	0.000
Algorithm Name_DBRF	0.708	0.113	0.092
Algorithm Name_EXTRA TREES	0.000	0.000	0.000

Shap

The results recive when pred_contribs (bool) parameter set to True in predict function of XGBoost:

AIP documentation: pred_contribs (bool) – When this is True the output will be a matrix of size (nsample, nfeats + 1) with each record indicating the feature contributions (SHAP values) for that prediction. The sum of all feature contributions is equal to the raw untransformed margin value of the prediction. Note the final column is the bias term.

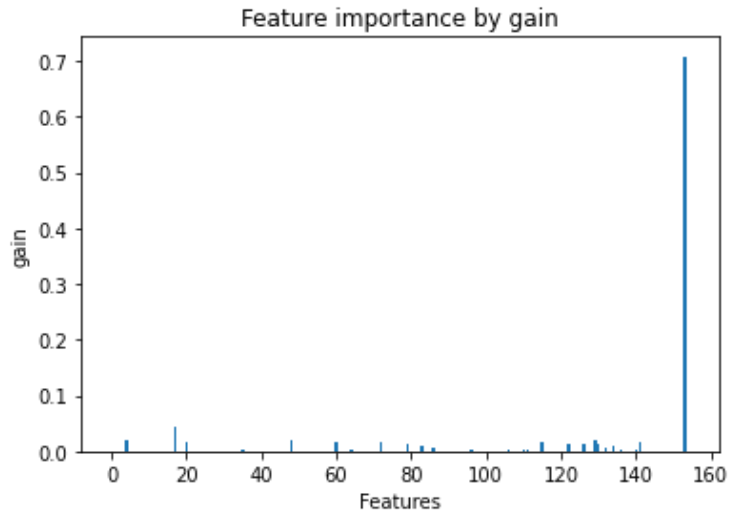
The table size we get is 156x300 so we present here only 5 rows because of place limitation.

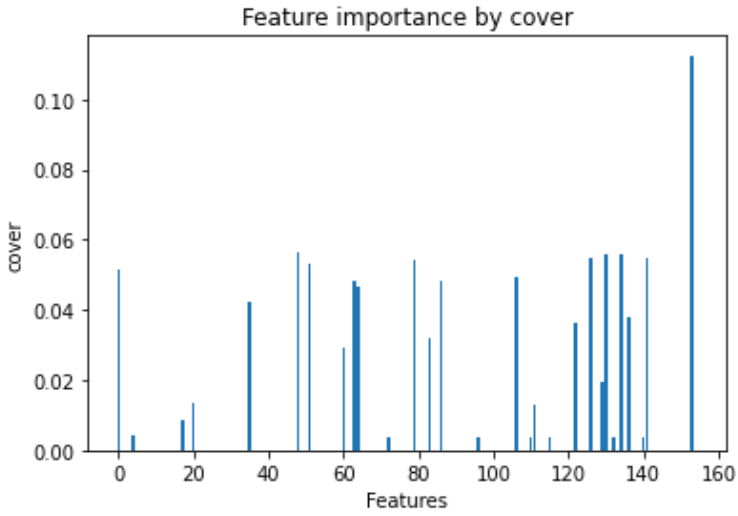
Feature/row	0	1	2	3	4
f1	-0.005318	0.0053184	0.0002296	-0.00023	0.0002296
f2	0	0	0	0	0
f3	0	0	0	0	0
f4	0	0	0	0	0
f5	0.0031528	-0.00326	-0.003621	0.0030066	0.0031528
f6	0	0	0	0	0
f7	0	0	0	0	0
f8	0	0	0	0	0
f9	0	0	0	0	0
f10	0	0	0	0	0
f11	0	0	0	0	0
f12	0	0	0	0	0
f13	0	0	0	0	0
f14	0	0	0	0	0
f15	0	0	0	0	0
f16	0	0	0	0	0
f17	0	0	0	0	0
f18	-0.001377	0.0013712	0.0028526	-0.002863	-0.005838
f19	0	0	0	0	0
f20	0	0	0	0	0
f21	0.0198216	-0.021377	0.0035279	-0.004143	-0.004985
f22	0	0	0	0	0
f23	0	0	0	0	0
f24	0	0	0	0	0
f25	0	0	0	0	0
f26	0	0	0	0	0
f27	0	0	0	0	0
f28	0	0	0	0	0
f29	0	0	0	0	0
f30	0	0	0	0	0
f31	0	0	0	0	0
f32	0	0	0	0	0
f33	0	0	0	0	0
f34	0	0	0	0	0
f35	0	0	0	0	0
f36	0.0012882	-0.001286	0.0012882	-0.001286	-0.026668
f37	0	0	0	0	0
f38	0	0	0	0	0
f39	0	0	0	0	0
f40	0	0	0	0	0
f41	0	0	0	0	0
f42	0	0	0	0	0
f43	0	0	0	0	0
f44	0	0	0	0	0
f45	0	0	0	0	0
f46	0	0	0	0	0

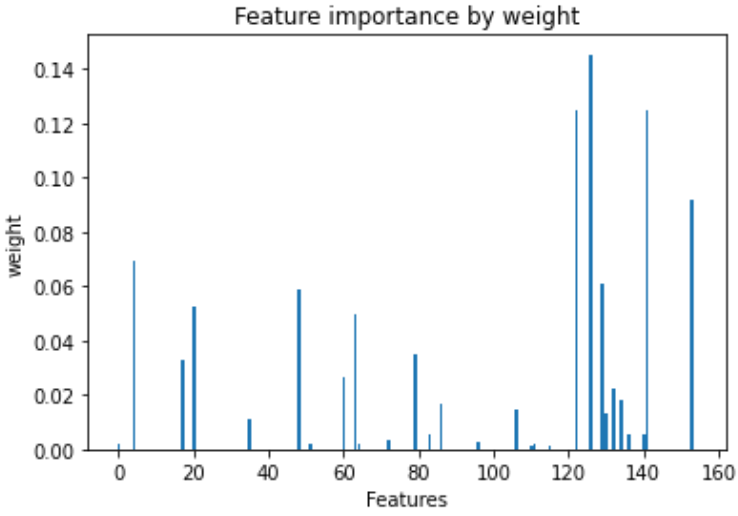
f47	0	0	0	0	0
f48	0	0	0	0	0
f49	0.0113617	-0.011351	0.0117448	-0.011734	0.0104344
f50	0	0	0	0	0
f51	0	0	0	0	0
f52	0.0002178	-0.000216	0.0002178	-0.000216	-0.005262
f53	0	0	0	0	0
f54	0	0	0	0	0
f55	0	0	0	0	0
f56	0	0	0	0	0
f57	0	0	0	0	0
f58	0	0	0	0	0
f59	0	0	0	0	0
f60	0	0	0	0	0
f61	0.0104309	-0.009193	0.003683	-0.003038	0.0026904
f62	0	0	0	0	0
f63	0	0	0	0	0
f64	0.0053809	-0.005377	0.0056736	-0.00567	0.0055188
f65	-0.004797	0.0047975	0.0002018	-0.000202	0.0002018
f66	0	0	0	0	0
f67	0	0	0	0	0
f68	0	0	0	0	0
f69	0	0	0	0	0
f70	0	0	0	0	0
f71	0	0	0	0	0
f72	0	0	0	0	0
f73	0.0003643	-0.000363	0.0001044	-0.000104	-0.000208
f74	0	0	0	0	0
f75	0	0	0	0	0
f76	0	0	0	0	0
f77	0	0	0	0	0
f78	0	0	0	0	0
f79	0	0	0	0	0
f80	0.0050485	-0.005051	0.0050485	-0.005051	0.0050485
f81	0	0	0	0	0
f82	0	0	0	0	0
f83	0	0	0	0	0
f84	0.0009372	-0.00104	0.000764	-0.000821	0.000764
f85	0	0	0	0	0
f86	0	0	0	0	0
f87	0.0028724	-0.00287	0.0028724	-0.00287	0.0016219
f88	0	0	0	0	0

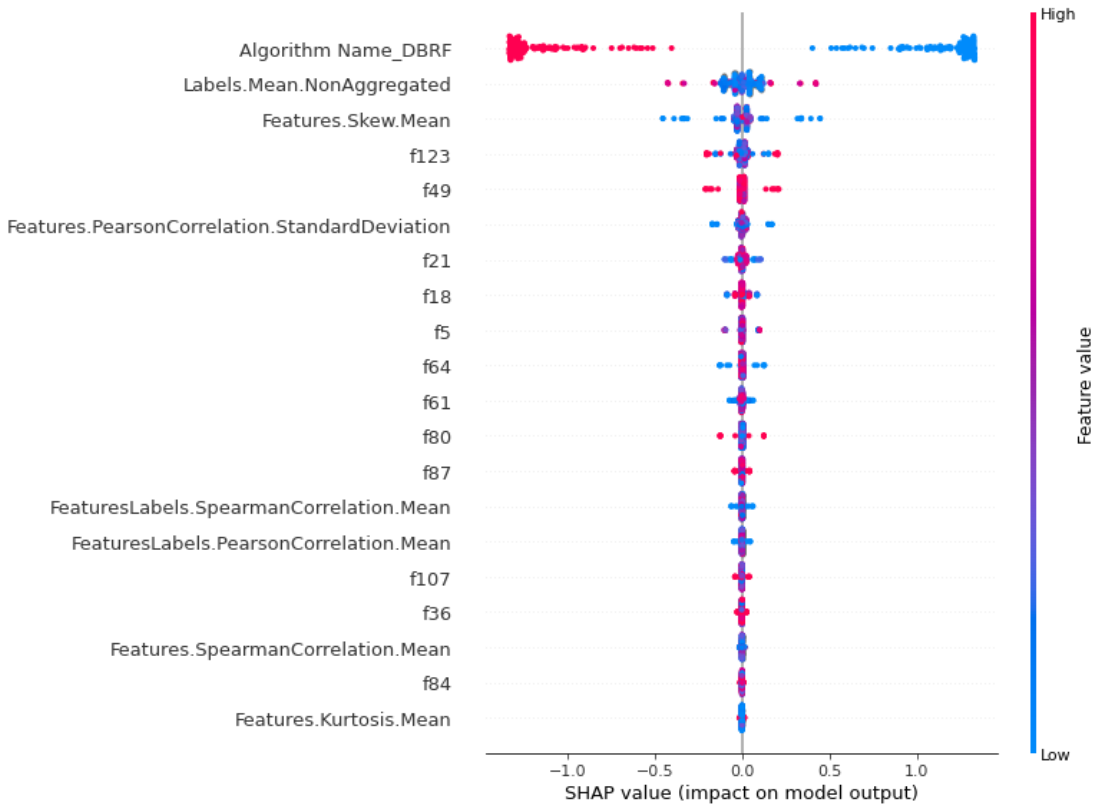
f89	0	0	0	0	0
f90	0	0	0	0	0
f91	0	0	0	0	0
f92	0	0	0	0	0
f93	0	0	0	0	0
f94	0	0	0	0	0
f95	0	0	0	0	0
f96	0	0	0	0	0
f97	2.14E-05	-0.000273	1.23E-05	9.17E-05	2.02E-06
f98	0	0	0	0	0
f99	0	0	0	0	0
f100	0	0	0	0	0
f101	0	0	0	0	0
f102	0	0	0	0	0
f103	0	0	0	0	0
f104	0	0	0	0	0
f105	0	0	0	0	0
f106	0	0	0	0	0
f107	0.0016351	-0.001635	0.0016351	-0.001635	0.0016351
f108	0	0	0	0	0
f109	0	0	0	0	0
f110	0	0	0	0	0
f111	-5.27E-05	-0.000105	5.14E-05	0.0001028	-5.27E-05
f112	0.0001168	-0.000117	7.58E-05	-7.60E-05	7.58E-05
f113	0	0	0	0	0
f114	0	0	0	0	0
f115	0	0	0	0	0
f116	0.0003365	0.0001446	-0.000113	-6.68E-05	9.35E-05
f117	0	0	0	0	0
f118	0	0	0	0	0
f119	0	0	0	0	0
f120	0	0	0	0	0
f121	0	0	0	0	0
f122	0	0	0	0	0
f123	0.0306541	-0.030693	0.0150259	-0.015041	-0.006445
f124	0	0	0	0	0
f125	0	0	0	0	0
Features.Skew.StandardDeviation	0	0	0	0	0
Features.Skew.Mean	0.0280739	-0.027572	0.0283225	-0.027864	0.0438055
Labels.Skew.NonAggregated	0	0	0	0	0
Features.PearsonCorrelation.Mean	0	0	0	0	0
Features.PearsonCorrelation.StandardDeviation	0.0223721	-0.022473	0.0035942	-0.003621	0.001355

FeaturesLabels.PearsonCorrelation.Mean	0.0017968	-0.001792	0.0017968	-0.001792	0.0017968
FeaturesLabels.PearsonCorrelation.StandardDeviation	0	0	0	0	0
Features.SpearmanCorrelation.Mean	0.0012105	-0.0012	0.0003548	-0.000349	0.0003521
Features.SpearmanCorrelation.StandardDeviation	0	0	0	0	0
FeaturesLabels.SpearmanCorrelation.Mean	0.0023379	-0.002334	0.0027836	-0.00278	0.0025579
FeaturesLabels.SpearmanCorrelation.StandardDeviation	0	0	0	0	0
Features.Kurtosis.Mean	0.0006503	-0.000649	0.0006503	-0.000649	0.0005073
Features.Kurtosis.StandardDeviation	0	0	0	0	0
Labels.Kurtosis.NonAggregated	0	0	0	0	0
Features.Mean.Mean	0	0	0	0	0
Features.Mean.StandardDeviation	-0.000278	0.0002744	-8.20E-05	7.96E-05	-0.000183
Labels.Mean.NonAggregated	-0.013637	0.0135649	0.0408328	-0.040732	0.0971886
Unnamed: 143	0	0	0	0	0
instances	0	0	0	0	0
dimensionality	0	0	0	0	0
instances_with_missing_val	0	0	0	0	0
ratio_of_discrete_features	0	0	0	0	0
ratio_of_numeric_features	0	0	0	0	0
nonzero_vals_cnt	0	0	0	0	0
missing values	0	0	0	0	0
attributes total	0	0	0	0	0
attributes categorical	0	0	0	0	0
attributes numerical	0	0	0	0	0
Algorithm Name_DBRF	1.2778382	-1.27776	1.324946	-1.324859	1.2529048
Algorithm Name_EXTRA TREES	0	0	0	0	0
bias term	-3.97E-05	-3.97E-05	-3.97E-05	-3.97E-05	-3.97E-05




Feature importance by cover
cover
0.00
0.02
0.04
0.06
0.08
0.10
0
20
40
60
80
100
120
140
160
Features


Feature importance by weight
weight
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
20
40
60
80
100
120
140
160
Features



CONCLUSIONS

In this work, we implemented the algorithm called "DBRF" and examined its performance on classification problems. we compared the results to the Extra Tree classifier (sklearn.ensemble.ExtraTreesClassifier) performance.

Its seems that the ExtraTreesClassifier "wins" in all parameters: this algorithm is fast and accurate in most of the cases. Our assumption is that the strength of the "DBRF" is in imbalanced data, so in regular data, it may not perform good as other classifiers.

In the Statistical significance of the results, Wilcoxon Signed-Rank Test, the p-value shows The results of those two algorithms evaluated on exactly the same data are statistically significantly different.

Learning-Meta model trained and tested on imbalanced data since the Extra Tree "wins" in most cases. And this is the reason the most importance feature is Algorithm name and the methods "cover" " weight" and " weight" have poor affect the results. Shap results also support this.

CITATIONS

" Deep Forest Regression for Short-Term Load Forecasting of Power Systems" wrote by LINFEI YIN, ZHIXIANG SUN, FANG GAO, AND HUI LIU and published in IEEE cite this paper. They don't relate to specific advantages or disadvantages of this paper.

The context is a part of works related to deep forest:

Quote from the paper:

"Deep forest regression is inspired by multi-grained cascade forest (gcForest), which can be called as deep forest… GcForest is a classification algorithm, which has been applied into numerous classification problems… As an ensemble classification method inspired by decision tree, the forecasting ability of the deep forest has a higher level as to that of DNNs. Furthermore, during the training process of a deep forest model, deep forest needs much lesser configured hyper-parameters than that of DNNs. GcForest and numerous improved gcForest algorithms have been applied to other applications. For instance,… ; dynamic boosted ensemble learning method has been proposed by Ren et al…

According to the experiments performed by Zhou and other improved gcForest algorithms, the major features of deep forest are listed as follows [42].

1) The deep forest with the default configuration of the hyper-parameters can be applied into numerous fields.

2) The structure of the deep forest could be paralleled trained with non-complex hyper-parameters.

3) The theoretical analysis of the deep forest is more easily than that of DNNs."