ML202 Final Assignment

Assaf Peleg

August 29, 2020

Note: I have used "we" along the report but it is a solo report.

1 Algorithm Name

MadaBoost: A Modification of **AdaBoost**

2 Reference

C. Domingo and O. Watanabe. Madaboost: A modification of Adaboost. In Proc. COLT 00, pages 180189, 2000

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

MadaBoost tries to fix some problems of **AdaBoost**, most notably: **AdaBoost** does not seem to be noise resistant. In addition **AdaBoost** cannot be used in the boosting by filtering framework. **We will test in this assignment only the noise resistant change(weighting change) on an altered sklearn AdaBoost subsampling framework implementation as unfortunately the running time of the test is too long to test also the filtering framework support.**

4 Short Description

MadaBoost attempts to solve these noise resistant problem by modifying the weighting system of **AdaBoost**. Instead of allowing the weights to be unbounded, **MadaBoost** uses a saturation bound that in practice can prevent overfitting.

5 Pseudo-Code

For the actual implementation and the code that tested the algorithms on the datasets, see FinalAssignment.py

Algorithm 1 MadaBoost

function TRAIN($S = (x_1, y_1), \ldots (x_N, y_N), T - number\ of\ rounds$)
  $D_{0_1} \leftarrow \frac{1}{N}, \ldots, D_{0_N} \leftarrow \frac{1}{N}$ ▷ Initial distribution of each sample
  for $i \leftarrow 1$ **upto** T **do**
    1. $S_i \leftarrow$ Select subset of S drawn according to $D_{(i-1)_1}$ to $D_{(i-1)_N}$
    2. $h_i \leftarrow$ Train the base classifier with S_i
    3. $\epsilon_i \leftarrow \sum_{t=1}^{N} D_{(i-1)_t} |h_i(x_t) - y_t|$ ▷ Errors of hypothesis i
    4. $\beta_i \leftarrow \sqrt{\epsilon_i/(1-\epsilon_i)}$
    for $k \leftarrow 1$ **upto** N **do** ▷ New distribution weights
      5. $w_{i_k} \leftarrow D_{0_k}(\prod_{1 \le j \le i} \beta_j^{cons(h_j, x_k, y_k)})$
    6. $W_i = \sum_{k=1}^{N} w_{i_k}$
    for $k \leftarrow 1$ **upto** N **do** ▷ Calculate probabilities from weights
      7. $D_{i_k} \leftarrow \frac{w_{i_k}}{W_i}$
      if $D_{i_k} > D_{0_k}$ **then** ▷ MadaBoost main change- saturation bound
        8. $D_{i_k} \leftarrow D_{0_k}$

function PREDICT(x)
  if $\prod_{i:h_i(x)=1} \beta_i \le \prod_{i:h_i(x)=0} \beta_i$ **then** ▷ Weighted majority voting
    return 1
  else
    return 0
function CONS(h, x, y) ▷ Customized indicator function
  if $h(x) = y$ **then**
    return 1
  else
    return -1

6 Algorithm Explanation

The algorithm input are the samples and the number of rounds(i.e the number of classifiers the ensemble will have). The flow of the algorithm is as follows:

1. First we initialize a distribution to sample the samples on each round.

2. Next we train T classifiers by first drawing a subset(with repetitions) of samples according to the distribution we initialized earlier.

3. Then, we train a classifier using that subset chosen.

4. Afterwards we calculate the errors and by using them we calculate a new distribution that will focus on the samples the classifier got wrong. The main change of **MadaBoost** is here, as we don't allow the weights probabilities to go over a saturation bound, that is be bigger than the initial distribution.

5. To classify the samples, we simply calculate a weighted majority voting of the classifiers.

7 Illustration

On each round, the algorithm trains a base classifier on samples sampled from a distribution. After each round, the weights probabilities of the distribution are changed to focus on the samples labeled wrong. Unlike **AdaBoost**, **MadaBoost** prevents overfitting to noise by using a saturation bound on the weights probabilities. See **Figure 1**.

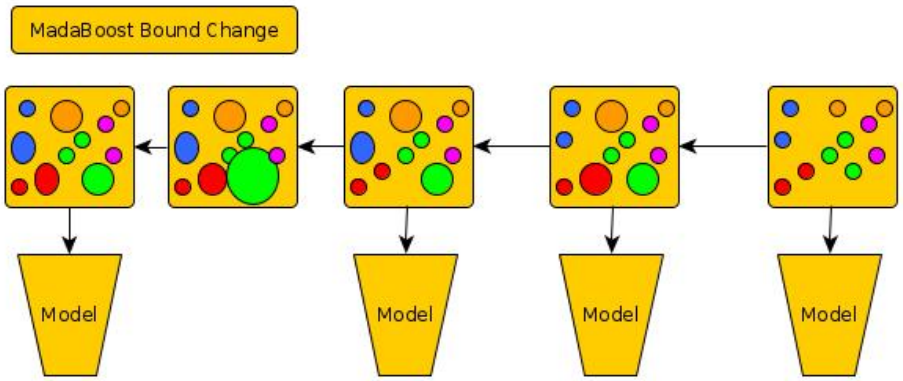

Figure 1: MadaBoost

We wrote a small notebook called **"MadaBoost Illustration"**(see MadaBoost Illustration.ipynb) to illustrate how the algorithm is bounding **AdaBoost** weights. To see full results see **weights_datset.csv**. In page 5 is a table for 6 iterations while we ran the algorithm for 100 iterations. The first 2 columns are the weights at the start. As we used 50 samples, the weights, or more accurately the probabilities that comes from the weights are 0.02 each. After that, on

iteration 2 we can see(marked in bold) that some samples were bounded. In the full csv we can spot some more bounded samples weights probabilities(see **weights_datset.csv**). Please note that 0 weights are not 0, only a very small number. Thus these samples might still be selected in the next iteration. **See table in the next page**.

Iteration 1	Bounded	Iteration 2	Bounded	Iteration 3	Bounded	Iteration 4	Bounded	Iteration 5	Bounded
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	0	0	0	0	0	0	0.0031	0.0031
0.02	0.02	0	0	0	0	0	0	0.0031	0.0031
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	0	0	0	0	0	0	0.0001	0.0001
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	**0.0678**	**0.02**	0	0	0.0013	0.0013	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0	0	0	0	0	0	0.0001	0.0001
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	0	0	0	0	0	0	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0	0	0	0
0.02	0.02	**0.0678**	**0.02**	0	0	0.0013	0.0013	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0	0	0	0	0	0	0.0001	0.0001
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0	0	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0 5	0	0	0	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0.0059	0.0059	0	0	0.0001	0.0001	0	0
0.02	0.02	0	0	0	0	0	0	0.0005	0.0005
0.02	0.02	0.0059	0.0059	0	0	0	0	0	0

8 Strengths

1) By using a saturation bound, **MadaBoost** is resistant to noise, unlike **AdaBoost**. Therefore, **MadaBoost** is unlikely to overfit like **AdaBoost**.

2) In another version of **MadaBoost**, it can be used in the **Filtering** framework and not just using the **Subsampling** framework. **AdaBoost** on the other hand supports only the **Subsampling** framework.

9 Drawbacks

1) As outlined in the paper, only certain type of noise **MadaBoost** is resistant to.

2) It's not clear how **MadaBoost** prevents under fitting due to class in-balance.

10 Experimental Results

We tested **MadaBoost** on 141 datasets from the 150 as some 9 datasets produced invalid roc curves due to problems with the predictions on these datasets. Unfortunately we couldn't understand what were the issues in time, as all of the other datasets did not cause any problems. We hope it is fine. We have chosen to test the algorithm against **LogitBoost** as the base-line. Moreover, We added **AdaBoost** to the test as its the the algorithm that **MadaBoost** attempts to fix. The test was preformed using a 3 fold cross validation to find the best hyper parameters and 10 fold cross validation for different test sets. The hyper parameters chosen were: DecisionTreeClassifier with different max depths in order to find the right bias, several different learning rates in order to find the right balance between speed of learning and accuracy, and ofcourse several different number of boosting rounds in order to optimize bias reduction. For **LogitBoost** the same applies only that it requires a DecisionTreeRegressor rather than a DecisionTreeClassifier. We chose to analyze the results using the roc auc measure. The results are included using three csv files:

1) **results.csv** is the file with all of the results.

2) **results_ranks.csv** is the file that allows to do statistical conclusions using the **Friedman** test. Calculated based on the roc auc measure.

3) **results_winner.csv** is the file to allow the creation of a meta learner. Calculated based on the roc auc measure.

The head(**some lines and not all columns**) of the files is given below:

results.csv

Full column name list of the file is: Dataset Name,Algorithm Name,Cross Validation [1-10],Hyper-Parameters Values,Accuracy,TPR,FPR,Precision,AUC,PR-Curve,Training Time(time.perf_counter()),Inference Time(time.perf_counter()

Dataset Name	Algorithm Name	Cross Validation [1-10]	Accuracy	Precision	AUC
led-display	MadaBoost	1	0.67	0.72	0.21
led-display	MadaBoost	2	0.68	0.68	0.18
led-display	MadaBoost	3	0.7	0.72	0.18
led-display	MadaBoost	4	0.74	0.76	0.13
led-display	MadaBoost	5	0.66	0.67	0.2
led-display	AdaBoost	1	0.66	0.7	0.22
led-display	AdaBoost	2	0.66	0.72	0.18
led-display	AdaBoost	3	0.6	0.63	0.22
led-display	AdaBoost	4	0.75	0.76	0.14
led-display	AdaBoost	5	0.65	0.66	0.2
led-display	LogitBoost	1	0.67	0.69	0.15
led-display	LogitBoost	2	0.72	0.73	0.13
led-display	LogitBoost	3	0.68	0.73	0.14
led-display	LogitBoost	4	0.77	0.8	0.13
led-display	LogitBoost	5	0.72	0.74	0.16

results_ranks.csv

Full column name list of the file is: Dataset Name,AdaBoost,LogitBoost,MadaBoost

Dataset Name	AdaBoost	LogitBoost	MadaBoost
led-display	1	3	2
cylinder-bands	1	2	3
wall-following	1	3	2
lupus	1	2	3
waveform	3	1	2

results_winner.csv

Full column name list of the file is: Dataset Name,Algorithm Name,Winner

Dataset Name	Algorithm Name	Winner
led-display	AdaBoost	True
led-display	MadaBoost	False
led-display	LogitBoost	False
cylinder-bands	AdaBoost	True
cylinder-bands	LogitBoost	False
cylinder-bands	MadaBoost	False
wall-following	AdaBoost	True
wall-following	MadaBoost	False
wall-following	LogitBoost	False
lupus	AdaBoost	True
lupus	LogitBoost	False
lupus	MadaBoost	False

To see which algorithm is statistically significant better than the others we will use the **Friedman** test using the **results_ranks.csv**:

First, we must calculate the mean ranks of each algorithm:

1) **Adaboost** Mean: 2.15
2) **LogitBoost** Mean: 2.02
3) **MadaBoost** Mean: 1.83

$$\chi_F^2 = \frac{12N}{k(k+1)}(\sum_j R_j^2 - \frac{k(k+1)^2}{4}) = \frac{12 \cdot 141}{3 \cdot 4}(2.15^2 + 2.02^2 + 1.8^2 - \frac{3 \cdot (4)^2}{4})$$

$$= 7.28$$

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} = \frac{(141-1)7.28}{141 \cdot 2 - 7.28}$$

$$= 3.7$$

With three algorithms and 141 data sets, F_F is distributed according to the F distribution with $(3-1) = 2$ and $(3-1)(141-1) = 280$ degrees of freedom. The critical value of $F(2, 280)$ for $\alpha = 0.05$ is 3.02, so we reject the null-hypothesis and therefore the algorithms are not equivalent.

Our next step would be to preform a post-hoc test between the algorithms. First, we'll need to calculate the critical difference for the two-tailed Nemenyi test.

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} = q_{0.05} \sqrt{\frac{3(3+1)}{6 \cdot 141}} = 2.343 \sqrt{\frac{3 \cdot 4}{6 \cdot 141}}$$

$$= 0.28$$

Note: $q_{0.05} = 2.343$ from the table of critical values for the two-tailed Nemenyi test.

If we examine the differences between the algorithms we'll see that **MadaBoost** is statistically significant better than **AdaBoost**(0.31 ranks difference). However, the difference betweens **MadaBoost** and **LogitBoost** is 0.19, which means that the post-hoc test did not help us to determine which of them is statistically significant better than the other. In addition, the difference between **AdaBoost** and **LogitBoost** is 0.12 which means that the test also did not help us to determine if **LogitBoost** is statistically significant better than **AdaBoost**. What is clear is that **MadaBoost** did outperform **AdaBoost**, and you should use it instead most of the time. However, it is not clear if it will be a safe bet against **LogitBoost** most of the time.

Our next step would be to find out on which datasets **MadaBoost** should be a safe bet. In order to do so, we built a meta learner to try and find patterns on which kind of datasets should you use this algorithm.

The code can be found in MetaLearner.ipynb.

The results of building the meta learner are included in the following csv files:

1) **meta_results.csv** is the file with all of the results.
2) **full_importance_df.csv** is the file that contains the feature importance extracted from xgboost.

The head(**some lines and not all columns**) of the files is given below:

meta_results.csv

Full column name list of the file is: Dataset Name Left Out,Cross Validation,Hyper-Parameters Values,Accuracy,TPR,FPR,Precision,AUC,PR-Curve,Training Time(time.perf_counter()), Inference Time(time.perf_counter()

Dataset Name Left Out	Cross Validation	Accuracy	Precision	AUC
acute-inflammation	1	0.67	0.33	1
acute-nephritis	2	1	1	1
analcatdata_asbestos	3	0.33	0.25	0.5
analcatdata_boxing1	4	0.33	0.25	0.5
analcatdata_broadwaymult	5	1	1	1
analcatdata_germangss	6	0.67	0.33	1
analcatdata_lawsuit	7	1	1	1
annealing	8	0.67	0.33	1
ar4	9	0.67	0.33	0.5
arrhythmia	10	0.33	0.25	0.5
audiology-std	11	1	1	1

full_importance_df.csv

Full column name list of the file is: weight,gain,cover,total_gain,total_cover

	weight	gain	cover	total_gain	total_cover
Algorithm Name=MadaBoost	252	0.071	7.091	17.97	1786.96
FeaturesLabels.PearsonCorrelation.StandardDeviation	70	0.67	9.10	46.80	637.31
Algorithm Name=AdaBoost	311	0.098	5.28	30.39	1641.74
f119	31	0.526	4.73	16.05	146.56
f120	24	0.55	5.22	13.27	125.24
f101	27	0.81	3.76	21.81	101.47
f110	35	0.37	6.7	12.88	234.26
nonzero_vals_cnt	55	0.75	7.90	41.10	434.68
Features.Mean.Mean	47	0.34	3.98	16.01	187.23
Features.Kurtosis.Mean	37	0.45	6.24	18.28	230.79
FeaturesLabels.SpearmanCorrelation.Mean	44	0.09	4.72	4.11	207.81
Features.Mean.StandardDeviation	70	0.65	7.56	45.77	529.33

In addition, we calculated some shap values to help us figure out when to use **MadaBoost**. Let's see some of them now:

First, let's see shap values for a prediction of row 6:

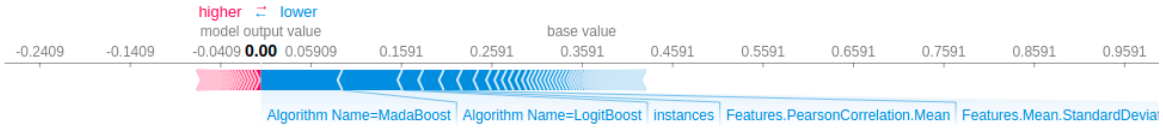

Figure 2: Prediction Plot of row 6

Let's see another:

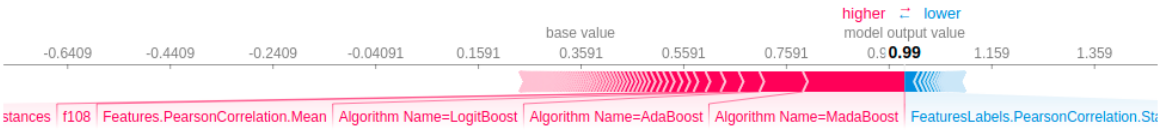

Figure 3: Prediction Plot of row 202

It's clear that the feature **MadaBoost** has the most influence on the prediction. We can see a confirmation for that in a summary plot:

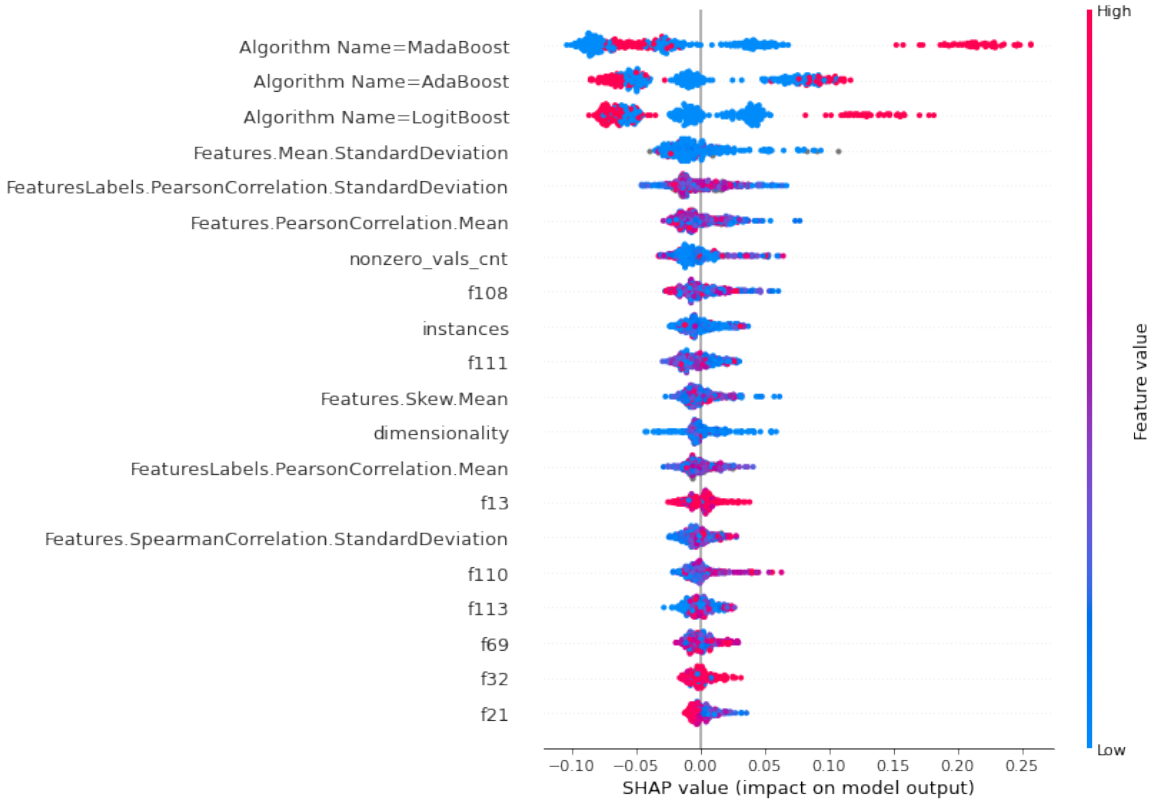

Figure 4: Summary Plot

We will expand on this later in the conclusions. Finally, let's examine more closely our **meta_results.csv**. The mean auc score of the entire cross validation is 0.57. If we calculate the mean when splitting the datasets by winner we will find that:

1) **Adaboost** Mean: 0.33
2) **LogitBoost** Mean: 0.53
3) **MadaBoost** Mean: 0.78

We can conclude that if the winner will be **MadaBoost**, it should be easy to figure it out. Otherwise, we can't really know if to choose **AdaBoost** or

LogitBoost. If we combine this with our **Friedman** test results, we can say that **MadaBoost** should be always preferred against **AdaBoost** but not always against **LogitBoost**. However, from the meta learner results, we now know that we can make this choice a more safe bet. Next, we'll dig in deeper in the conclusions to figure out on which datasets should you most certainty choose **MadaBoost**.

11 Conclusions

First, we can conclude that adding a saturation bound on the weight of each sample does help to avoid overfitting. However, On some datasets **AdaBoost** and **LogitBoost** does seem to preform better- probably due to under fitting due to class in-balance. Using the meta learner we built, we can try to figure out when one might prefer to use **MadaBoost** over the others. **For full details of the meta learner, the shap values, the meta learner csv results and the feature importances see: MetaLearner.ipynb(includes shap values), meta_results.csv, full_importance_df.csv respectively.**

First, let's try to use the summary plot:

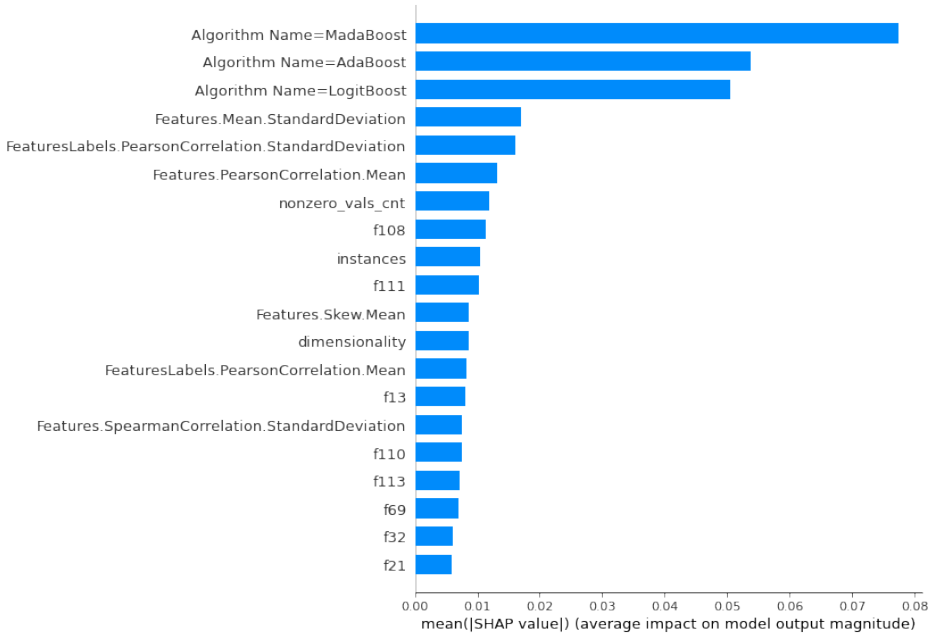

Figure 5: Summary Plot

From the summary plot, it is clear that if the given algorithm is **MadaBoost** it is fairly easy to determine if it will preform better or worse. Let's try to use some other measures to see if we can determine how the choice could be made.

The next interesting feature is Standard Deviation as its impact is high and it directly linked to samples noise which **MadaBoost** attempts to address:

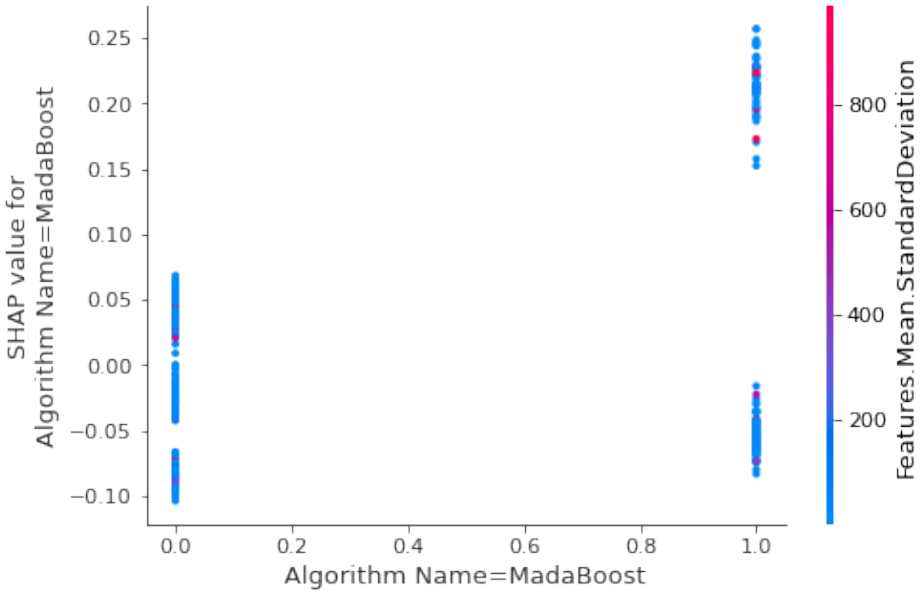

Figure 6: Dependence plot

We can see from the plot that really high and low standard deviation for dataset tested on **MadaBoost** affects greatly it's ability to outperform **AdaBoost** and **LogitBoost**.

⇒ As seen from the shap values, **MadaBoost** is significantly better than **AdaBoost** and **LogitBoost** on datasets with high or low standard deviation of its features. Thus, these are the types of data that you should use **MadaBoost** on.

On a final note, we did find that **MadaBoost** fixes **AdaBoost** overfitting issues and is statistically significant better. That is it will preform better on most datasets. However, you should check the standard deviation of the dataset to make sure you are choosing the algorithm correctly. On the other hand, we couldn't show that **MadaBoost** will preform better than **LogitBoost**, but as with **AdaBoost** you should choose it if the standard deviation of the datasets is somewhat high or low.

12 Citations

C. Domingo and O. Watanabe. Madaboost: A modification of Adaboost. In Proc. COLT 00, pages 180189, 2000

Statistical Comparisons of Classifiersover Multiple Data Sets

SKLearn AdaBoost Implementation

A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting