Assignment Report Group 26

Aleksandra Baumgart, Kerem Bildiren, Mireia Planas Lisbona

Leiden Institute of Advanced Computer Science, The Netherlands

1 Introduction

In this assignment, we will be implementing an ensemble learning algorithm. First, we will implement a weak base learner (linear halfspaces). It will be used as our basis for an ensemble learning model using Adaboost.

The idea behind AdaBoost is to train a lot of weak classifiers and use that to make a strong assumption (a voting mechanism). The prediction of weak classifiers is binary. Each of the classifiers has its own weight which will affect the prediction accordingly. The new classifier is more sensitive to wrong observations but it is not as sensitive to the correct ones.

After coding the models, we will run performance tests on the algorithms and compare accuracy scores throughout different experiments.

2 Background

The main model explored in this assignment is called Adaboost. This is a boosting procedure that improves the performance of weak learners by sequentially applying them to weighted data, based on which samples are more informative.

The base learner used in this assignment will be linear half spaces. This is a supervised learning model that creates a linear decision boundary between two classes of labels. The linear half spaces model returns the label [4]:

$$y(x) = sign(< w, x > +b)$$

To train the model we will use the batch perceptron algorithm with a learning rate decreasing linearly.

Next, we will construct the Adaboost model. Adaboost is short for Adaptative Boosting and it describes a technique formulated by Yoav Freund and Robert Schapire [3]. It works by training new weak learners on samples that are more difficult to classify. In the beginning, we assign uniform probabilities to all the samples of the training set. At each boosting iteration, we sample a subset of the training examples according to their probability distribution and we train a weak learner on these samples. Next, we evaluate the new weak classifier to obtain its performance. Then, we update the sample weights based on the prediction of the new classifier on each specific sample. If the weak model correctly classifies a sample, we will decrease its probability; if it fails, we will increase it. After repeating this procedure for as many weak learners as we want, we can produce a prediction by weighting the individual predictions of the weak learners (with weights inversely proportional to the model's errors).

The pseudo code of the method used is the following [2]:

Algorithm 1: Adaboost

Initialize the sample weights to $w_i = 1/N,\ i = 1, 2, ..., N$
for $m \leftarrow 1, ...M$ **do**
(a) Sample with repetition a p% of the training data with probability given by w_i.
(b) Fit a classifier $G_m(x)$ to the sampled data.
(c) Compute

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

(d) Compute $\alpha_m = 0.5 \cdot \log \frac{1-err_m}{err_m}$
(e) Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot (I(y_i \neq G_m(x_i)) - I(y_i = G_m(x_i)))\right]$ and normalize the weights.
return $G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

The variable α_m measures the performance of a base learner. Using the formula [1]:

$$\alpha_m = 0.5 \cdot \log \frac{1 - err_m}{err_m}$$

we get zero performance if the total error is 0.5 and $+\infty$ or $-\infty$ if the error is 0 or 1 respectively. In practice, a small error term is added to avoid divergence.

3 Results[1]

To determine the performance of our algorithm we applied several tests using the provided datasets. Each dataset was randomly split into training and test sets with the test set being 20% of the whole dataset. After preparing the data, we proceeded to train the model on the training set. Finally, we made predictions and calculated the accuracy scores on the test set.

3.1 Accuracy and decision regions of the base learners

The first thing we inspected about our models is the decision boundary created in each case. Figure 2 shows the decision regions of the base learners for datasets 1, 2 and 3. We can observe that the first dataset is linearly separable, so the base learner (linear halfspaces) is able to tell the two classes apart. However, datasets 2 and 3 are not linearly separable, which makes it impossible for the base learner to succeed. By averaging the results over 50 runs we obtained the test accuracies shown in Table 1.

3.2 Accuracy and decision regions of Adaboost

Figure 3 shows the decision boundaries created by the Adaboost model with different numbers of estimators. We see that the model with 10 estimators fails to classify some of the test samples

[1] Unless otherwise stated the experiments have been performed with an initial learning rate of $\mathbf{1e-2}$, a maximum number of iterations of **1000** and a sampling percentage for Adaboost of **0.1**. The weights of the base learner have been initialized using a standard normal distribution.

Dataset 1	Dataset 2	Dataset 3	Dataset 4
99.6%	50.5%	47.9%	48.6

Table 1: Average test accuracies of the base learners over 50 runs.

because it might be overfitting. However, the models with 100 and 1000 estimators have perfect accuracy and the boundary is more smooth the more estimators we have.

In Figure 4 and Figure 5 we can see that, for non linearly separable datasets, the increase in the number of estimators, increases the accuracy. The more estimators we use in Adaboost, the more complex decision regions we can create, thus solving the problem of the data not being linearly separable. For example for dataset 2, the accuracy with 10 estimators is around 65% while the accuracy with 1000 estimators is around 98%. For dataset 3 the accuracy with 10 estimators is around 56% while the accuracy with 1000 estimators is around 97%. In conclusion, it is safe to say that with the growing number of estimators the accuracy increases.

In our experiments, we did not come across the problem of overfitting. However, we know that a high number of estimators can create complex models that could stumble upon overfitting.

3.3 Learning rate analysis for the base learner

In this section, we want to test how stable the base algorithm is to the choice of learning rate. We chose logarithmic spaced learning rates and measured the test accuracy of the weak learners as showed in Figure 1(a).

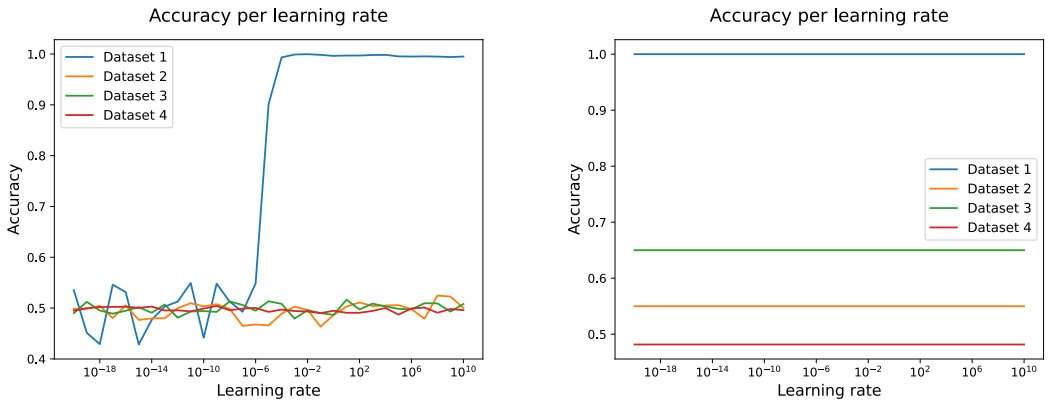

Fig. 1: (a) Test accuracy per learning rate of the base algorithm for weights initialized from a standard normal distribution. Average over 50 runs for each setting. (b) Test accuracy per learning rate of the base algorithm for weights initialized as zeros. Average over 50 runs for each setting.

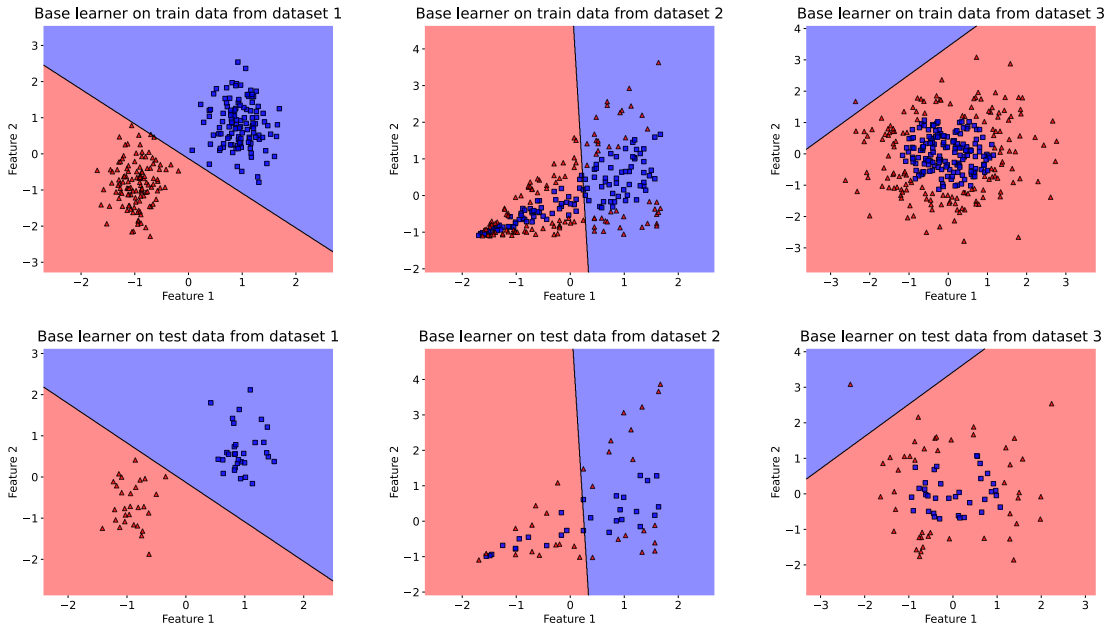

Fig. 2: Base learner decision regions applied to train and test data for datasets 1, 2 and 3.

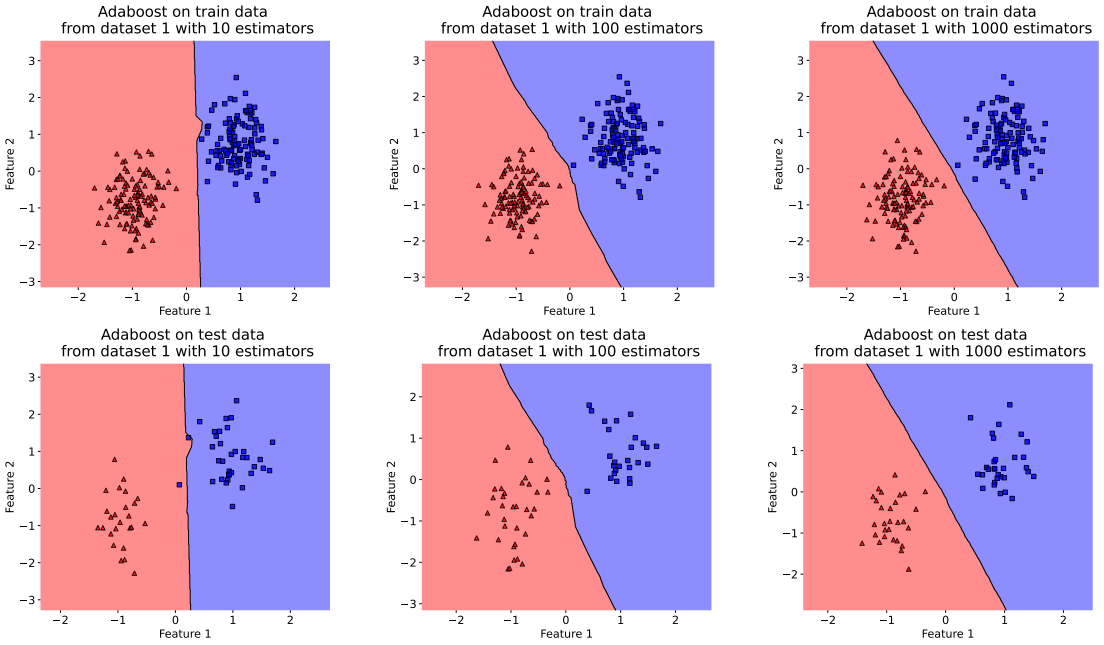

Fig. 3: Adaboost decision regions for 10, 100 and 1000 estimators applied to train and test data for dataset 1.

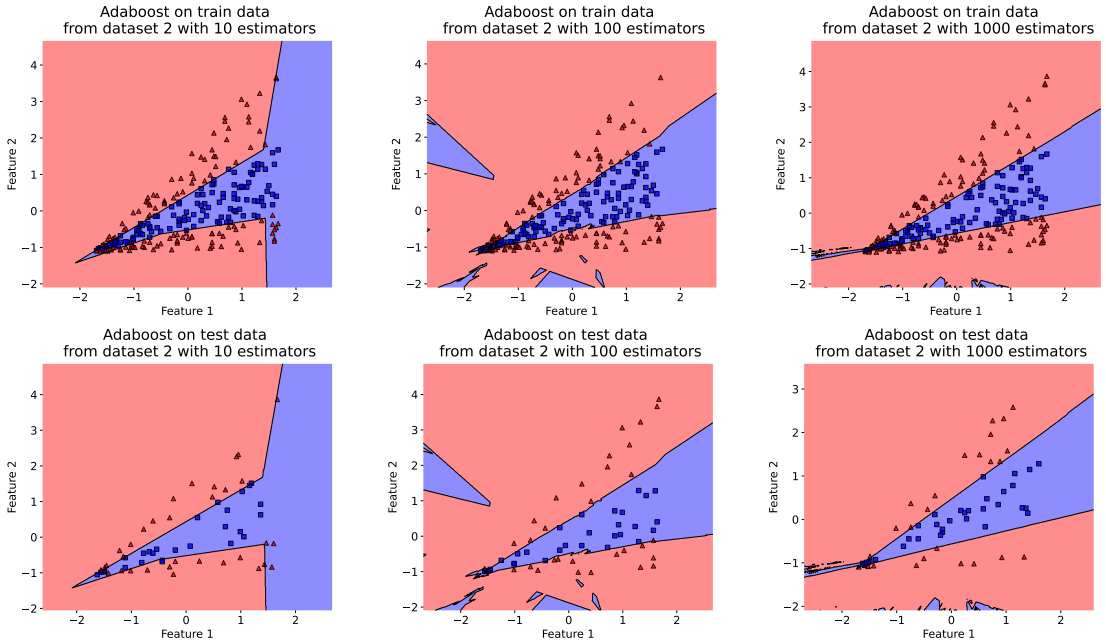

Fig. 4: Adaboost decision regions for 10, 100 and 1000 estimators applied to train and test data for dataset 2.

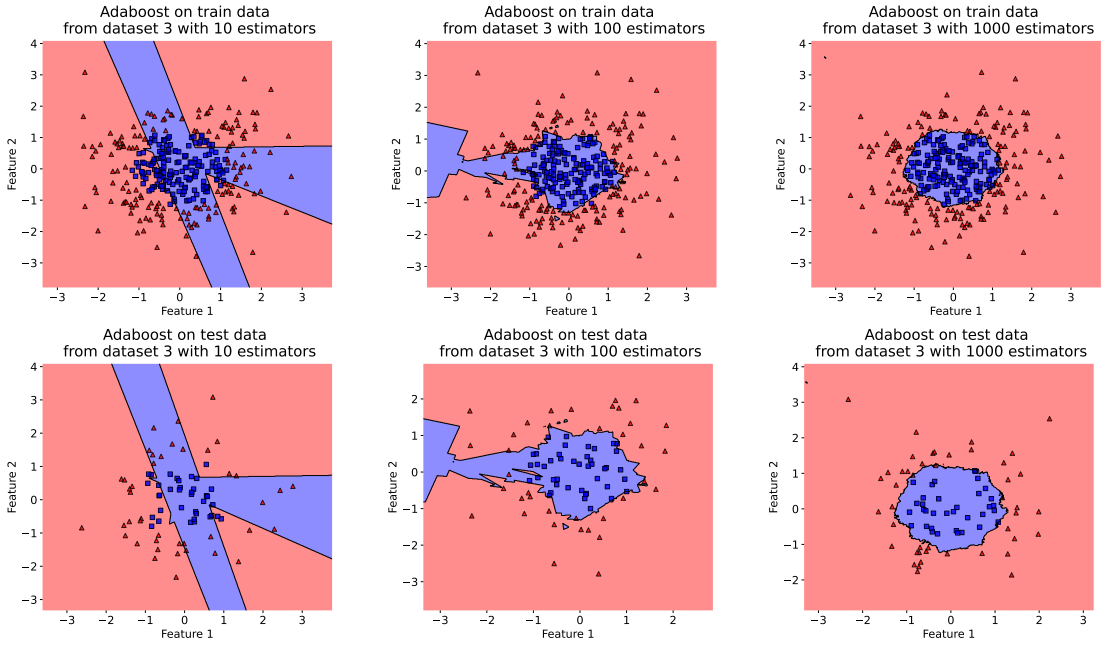

Fig. 5: Adaboost decision regions for 10, 100 and 1000 estimators applied to train and test data for dataset 3.

We can see that the accuracy only increases for the first dataset as the learning rate increases. The weak learner is not capable to perform well for other complex datasets. In Figure 1(b), the weights are initialized as zeros. As we can see, the performance is constant. Since the weights start at 0 the learning rate only affects to the norm of the weight vector, and not to the direction of the plane separating the classes. Therefore, the learning rate has no effect on the performance of the algorithm. For machine learning models that learn through gradient descent, initializing the weights to zero is usually a bad idea since all the outputs will start at zero and all the gradients will be the same (no symmetry breaking).

3.4 Analysis of the number of estimators and sampling percentage for Adaboost

To explore the behavior of the boosting algorithm, we tested the changes in accuracy with different numbers of estimators. Then, we tested the changes in performance for different sampling percentages. We chose 10 values between 0 and 1 and ran the experiment 5 times to obtain the average as we used the number of estimators equal to 1000 and the running time is too high to average over 50 runs.

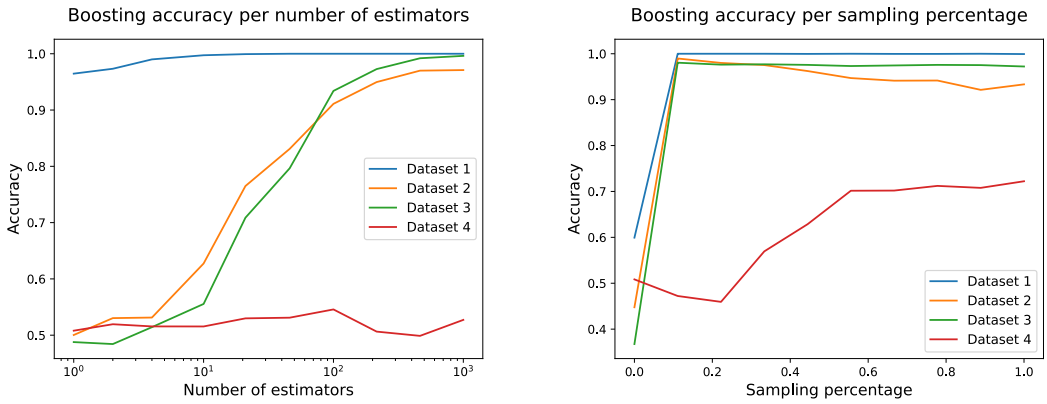

Fig. 6: (a) Test accuracy of Adaboost per number of estimators. Average over 50 runs for each setting. (b) Test accuracy of Adaboost with 1000 estimators per sampling percentage. Average over 5 runs for each setting.

As we mentioned above, with the growing number of estimators, the accuracies increase. This is visible in the Figure 6(a). Nonetheless, it is not true for dataset 4 as it contains too many features. This increased dimensionality leads to noisiness in the data and makes it very difficult for the model to learn.

In the Figure 6(b) we compared the influence of different sampling percentages on the accuracy. For dataset 4, the most complex one, it is visible that the model needs more data in the sample population to be trained properly. For other datasets the increase of value of the sampling percentage does not seem to be significant, however there is a slight drop in accuracy for dataset 2.

3.5 Analysis of the sample distribution update rule

In the Adaboost model there exist several methods for updating the sampling probability distribution after training each base learner. In this section we want to explore some of these updating rules in order to study their impact on the performance of the ensemble.

We chose three different update rules which are described as follows:

- **Update rule 1:** In this rule we increase the weight of the misclassified samples and decrease the weights of correctly classified ones according to the pseudo code in section 2:
$$w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot (I(y_i \neq G_m(x_i)) - I(y_i = G_m(x_i)))\right]$$
- **Update rule 2:** In this case, the formula for updating the weights is the same, but the weights are restarted to $1/N$ after each boosting iteration. Hence, the probability distribution only depends on the last base learner that was trained.
- **Update rule 3:** In this method the update of the weights is asymmetric. We increase the probability of misclassified samples but we don't update the weight of correctly classified samples (only through normalization). The update formula is as follows:
$$w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I(y_i \neq G_m(x_i)))\right]$$

In Figure 7 we show the performance during training for these three methods. The x axis shows the Adaboost iteration (each iteration adds a base learner) and the y axis shows the test accuracy.

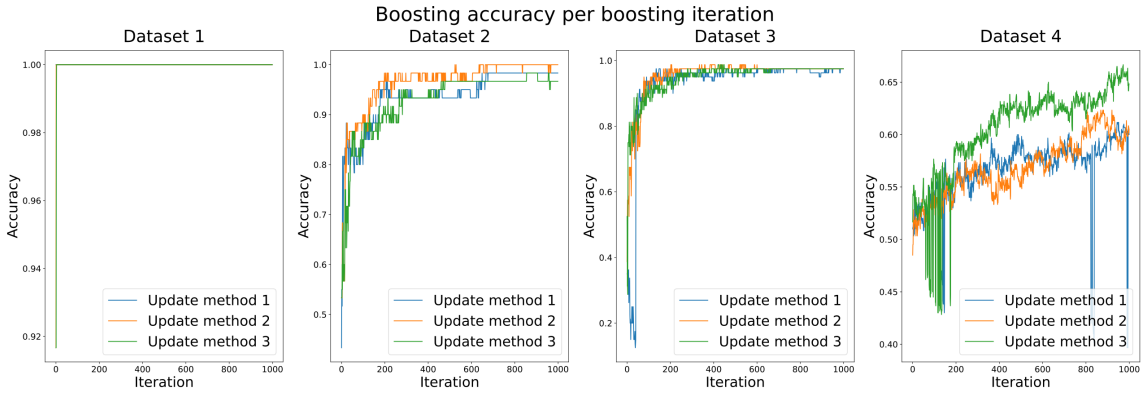

Fig. 7: Test accuracy at each iteration of Adaboost for different sample distribution update rules.

As we see in the previous figure, all the methods have perfect performance on the first dataset. On the second dataset the second method learns slightly faster and the first and third update rule have similar performance. On the third dataset we don't see any significant differences. Finally, in the fourth dataset, the third method outperforms the others in higher iterations but the results are noisy due to the complexity and high dimensionality of the data, so we can't get to a solid conclusion. In general, we would say that there is no clear benefit on using one of the methods above the others as they all perform similarly.

4 Extending to multiclass classification

As our model was trained for binary classification, we can also consider extending the problem to deal with more than two classes. To do that we could transform the problem into several binary problems using a one vs. all strategy. If we were to implement that we would have to also change the weight updating rule so it takes into consideration the number of classes. The changed formula could look as follows according to SAMME algorithm [5]:

$$\alpha_m = \log \frac{1 - err_m}{err_m} + \log(K - 1),$$

where K indicates the number of classes.

5 Conclusions

We managed to successfully implement an ensemble learning algorithm. By observing the decision boundaries created by Adaboost we realized how powerful ensemble learning can be, since it is able to solve complex classification regions using only weak learners. In the results of our tests and experiments, we can clearly notice that the base learner only performs well for the first dataset. We have also seen that with the first three datasets, an increase in the number of estimators increases the accuracy. In the analysis of the learning rate for the first dataset, we found that the base model learns a good classification boundary for high learning rates. We have studied the effect of initializing the weights of the base learner to zero and why it is not a good practice. Finally, we experimented with different update rules for the sampling probabilities in Adaboost and we found no noticeable differences. All in all, we have gained a deep understanding of how boosting algorithms work and why they have such a promising performance.

References

1. Implementing the adaboost algorithm from scratch, https://www.kdnuggets.com/2020/12/implementing-adaboost-algorithm-from-scratch.html
2. Corrales Cano, A.: Adaboost from scratch (Apr 2021), https://towardsdatascience.com/adaboost-from-scratch-37a936da3d50
3. Freund, Y., Schapire, R.E.: A desicion-theoretic generalization of on-line learning and an application to boosting. In: Vitányi, P. (ed.) Computational Learning Theory (1995)
4. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press (2014), https://books.google.nl/books?id=ttJkAwAAQBAJ
5. Veronica, A.: Understanding adaboost and scikit-learn's algorithm: - medium, https://medium.datadriveninvestor.com/understanding-adaboost-and-scikit-learns-algorithm-c8d8af5ace10