Mathematical foundations of Machine Learning 2024 – lesson 5

Shai Dekel

TEL AVIV UNIVERSITY

Gradient boosting

AdaBoost – binary classification case

We have a dataset $\{(x_i, y_i)\}_{i\in I},\ x_i \in \mathbb{R}^n, y_i \in \{-1,1\}$

Boosting: sequentially, at the m-th iteration, apply a "weak classifier" (e.g. logistic regression, SVM) G_m on weighted versions of the training set (or subset). For example:

Weighted NLL loss $$-\frac{1}{\sum_{i\in I} w_i}\left\{\sum_{y_i=1} w_i \log h_{\theta_m}(x_i) + \sum_{y_i=-1} w_i \log\left(1-h_{\theta_m}(x_i)\right)\right\}$$

$$h_{\theta_m}(x) := \frac{1}{1+e^{-\left(\langle \vec{\beta}_m, x\rangle + \beta_{0,m}\right)}} \qquad G_m(x) = \operatorname{sgn}\left(2h_{\theta_m}(x) - 1\right)$$

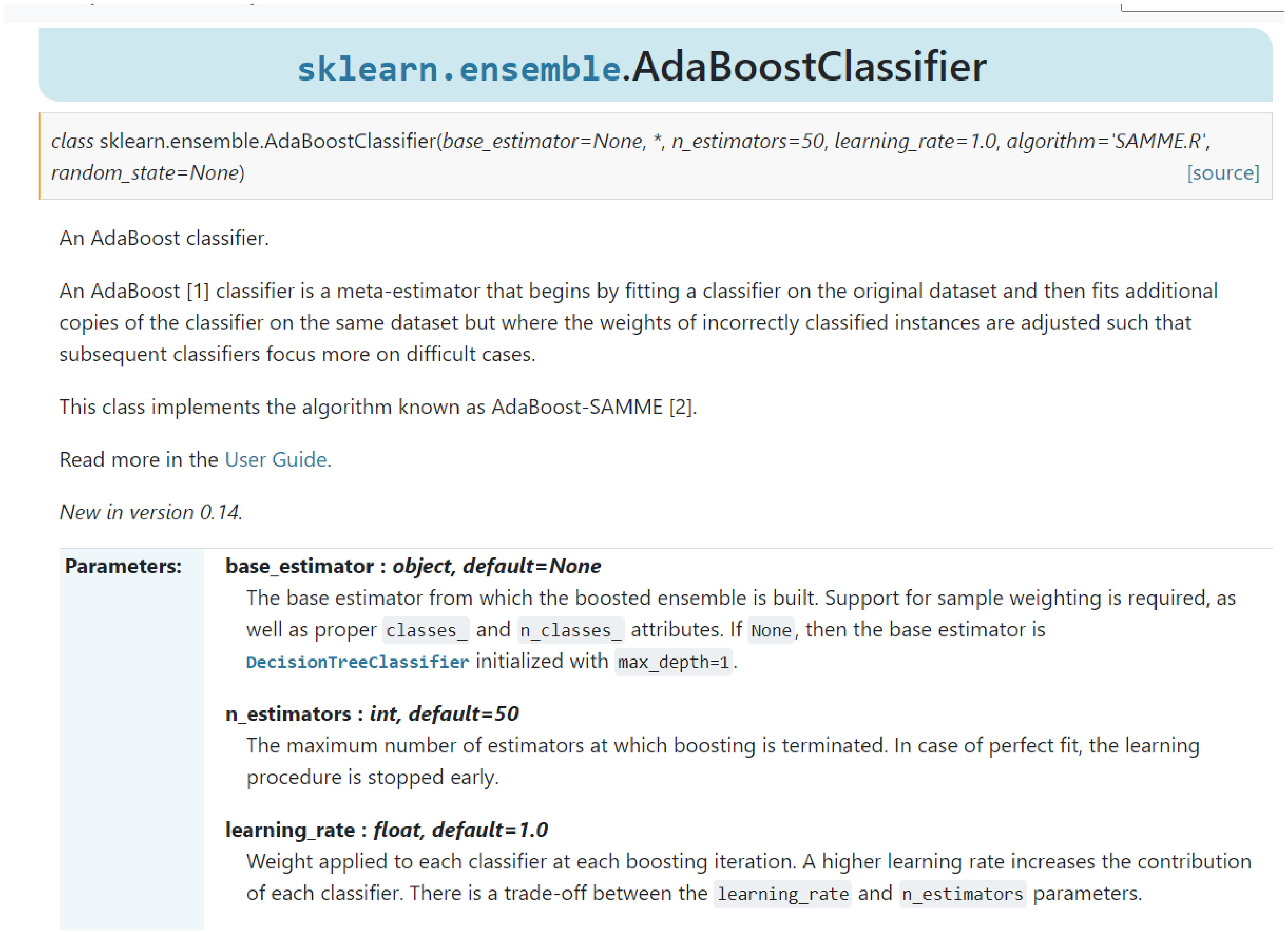
Create an aggregated "strong classifier" $G(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$

Algorithm 10.1 *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.
```

$$\mathrm{err}_m < 0.5 \Rightarrow \frac{1-\mathrm{err}_m}{\mathrm{err}_m} > 1$$

$$\Rightarrow \alpha_m > 0$$



sklearn.ensemble.AdaBoostClassifier

class sklearn.ensemble.AdaBoostClassifier(*base_estimator=None*, ***, *n_estimators=50*, *learning_rate=1.0*, *algorithm='SAMME.R'*, *random_state=None*) [source]

An AdaBoost classifier.

An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

This class implements the algorithm known as AdaBoost-SAMME [2].

Read more in the User Guide.

New in version 0.14.

Parameters:

base_estimator : *object, default=None*

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier` initialized with `max_depth=1`.

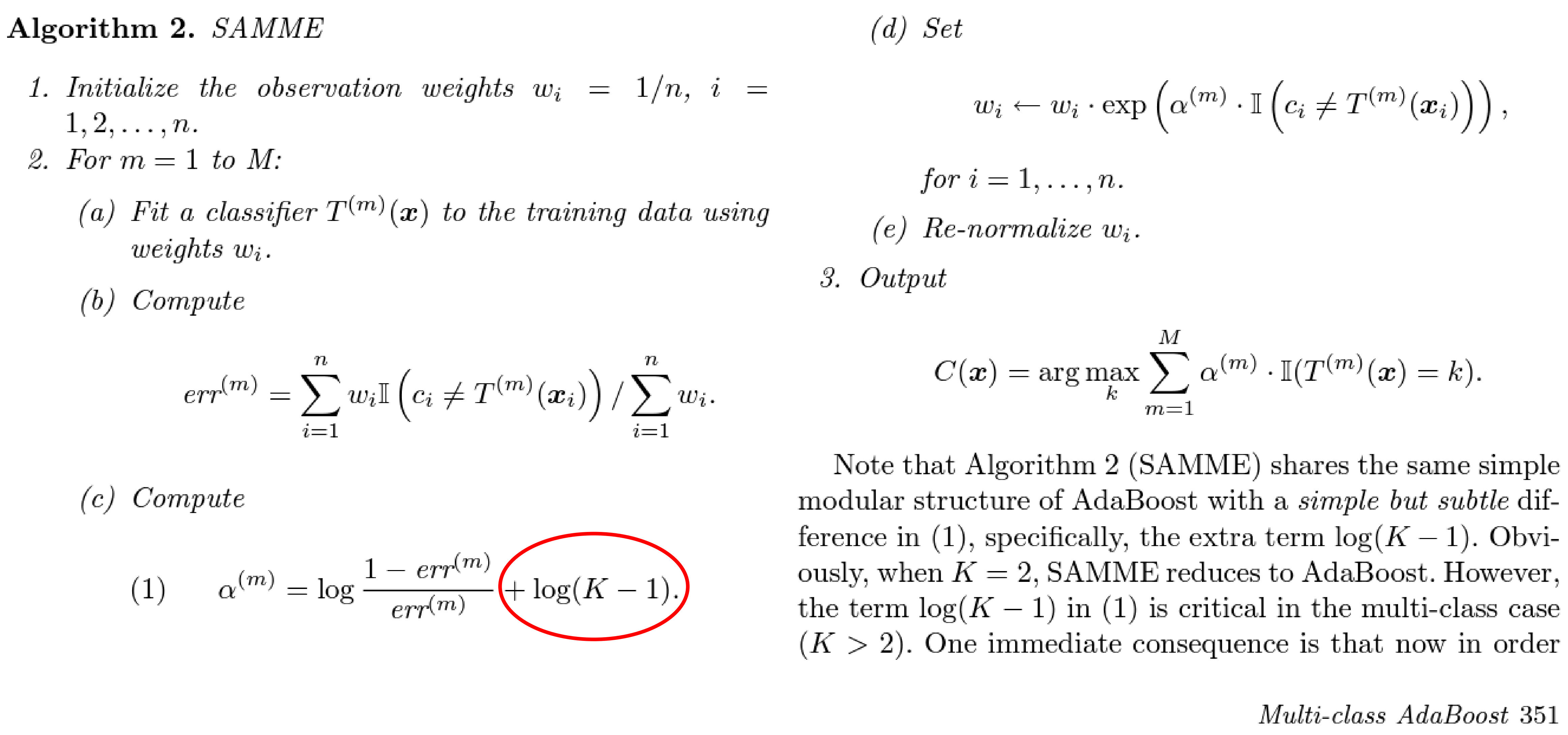
n_estimators : *int, default=50*

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate : *float, default=1.0*

Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators` parameters.

AdaBoost-SAMME (multi-class case)



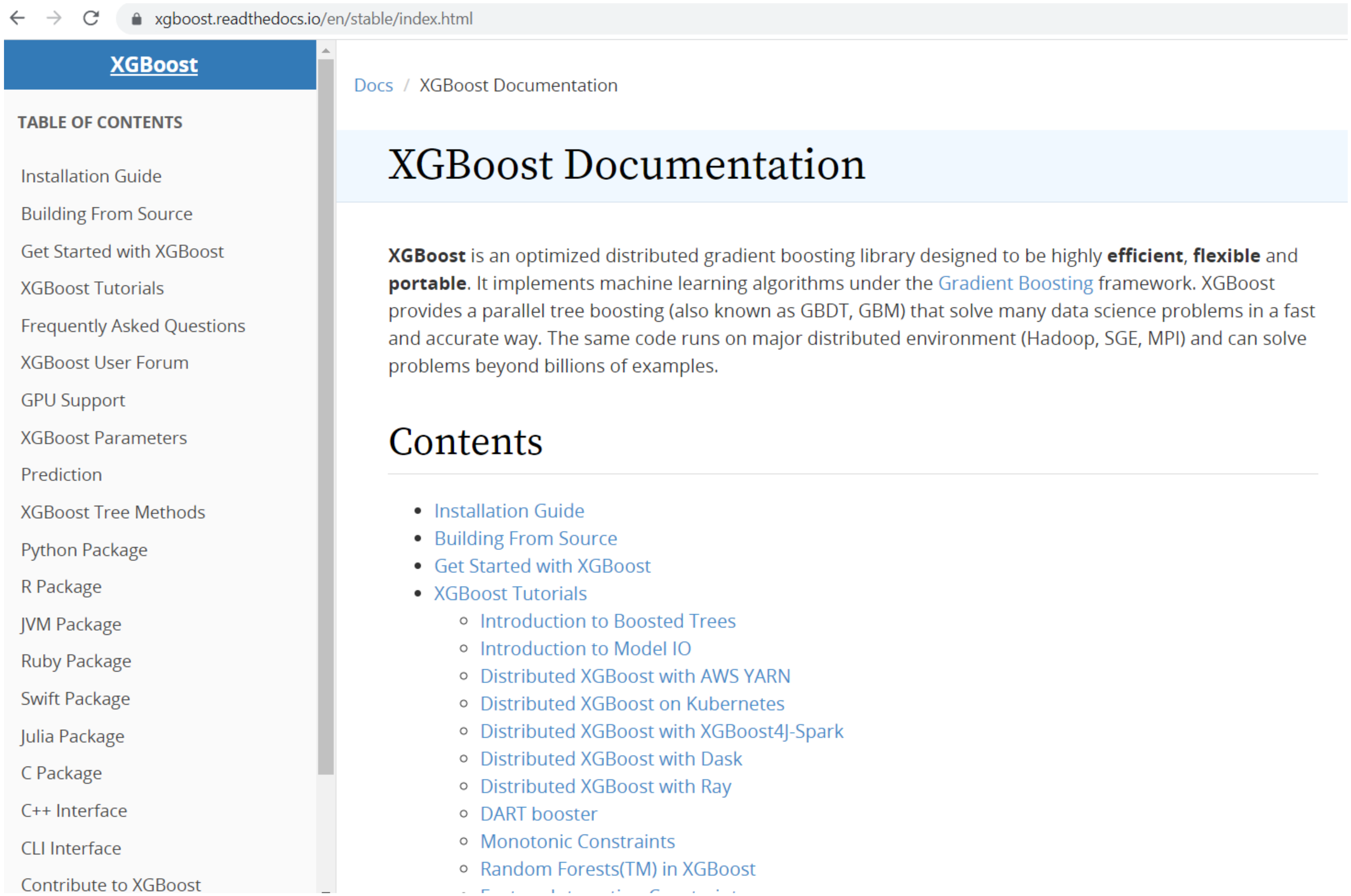
Algorithm 2. *SAMME*

```
1. Initialize the observation weights $w_i = 1/n$, $i = 1, 2, \ldots, n$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $T^{(m)}(\boldsymbol{x})$ to the training data using weights $w_i$.
    (b) Compute
        $$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$
    (c) Compute
        (1)    $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K-1).$$
    (d) Set
        $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$
        for $i = 1, \ldots, n$.
    (e) Re-normalize $w_i$.
3. Output
        $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$
```

Note that Algorithm 2 (SAMME) shares the same simple modular structure of AdaBoost with a *simple but subtle* difference in (1), specifically, the extra term $\log(K-1)$. Obviously, when $K = 2$, SAMME reduces to AdaBoost. However, the term $\log(K-1)$ in (1) is critical in the multi-class case $(K > 2)$. One immediate consequence is that now in order

*J. Zhu, H. Zou and S. Saharon, Multi-class Adaboost, statistics and its interface 2 (2009), 349-360.

XGBoost



xgboost.readthedocs.io/en/stable/index.html

XGBoost

TABLE OF CONTENTS

Installation Guide
Building From Source
Get Started with XGBoost
XGBoost Tutorials
Frequently Asked Questions
XGBoost User Forum
GPU Support
XGBoost Parameters
Prediction
XGBoost Tree Methods
Python Package
R Package
JVM Package
Ruby Package
Swift Package
Julia Package
C Package
C++ Interface
CLI Interface
Contribute to XGBoost

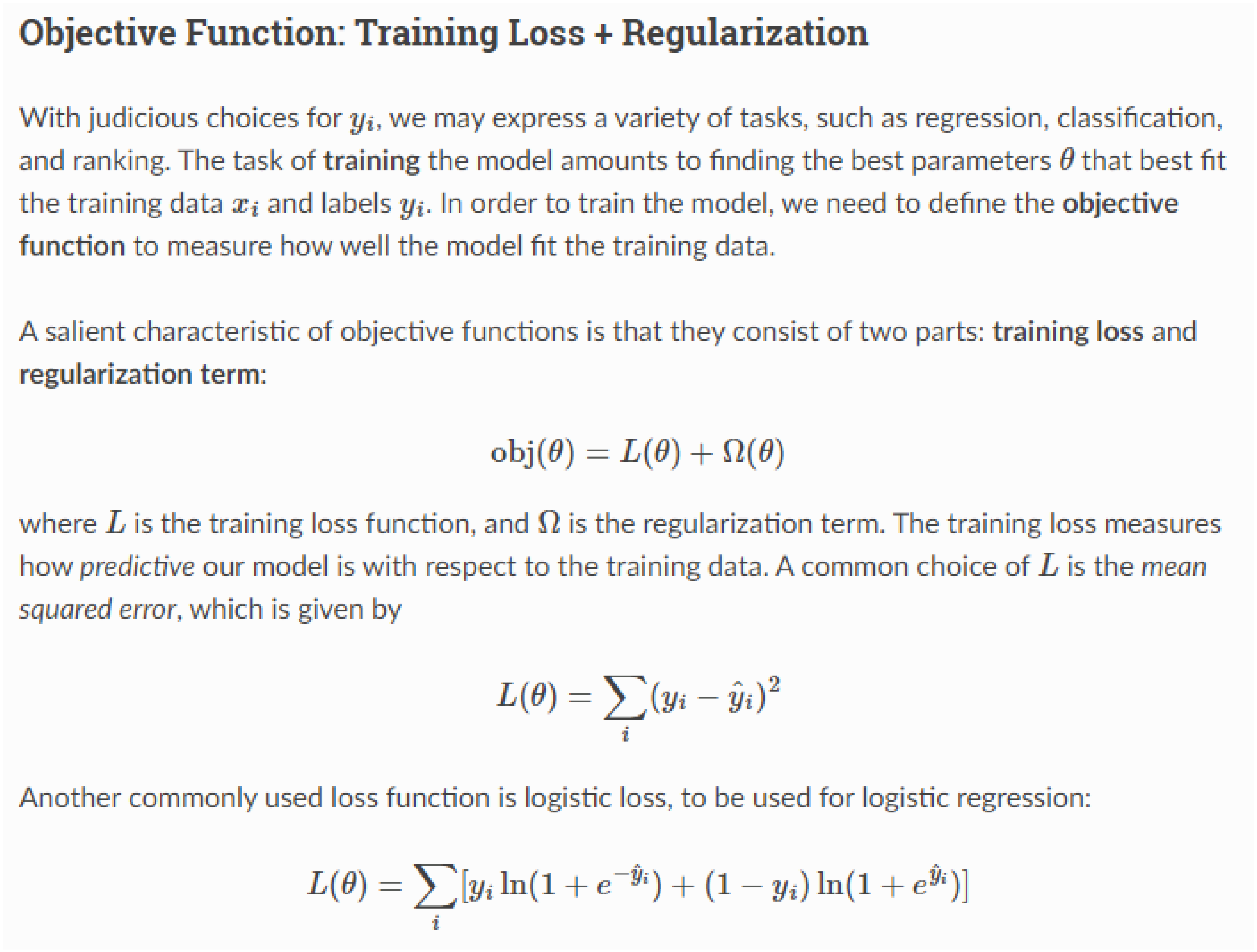
Docs / XGBoost Documentation

XGBoost Documentation

XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Contents

- Installation Guide
- Building From Source
- Get Started with XGBoost
- XGBoost Tutorials
  - Introduction to Boosted Trees
  - Introduction to Model IO
  - Distributed XGBoost with AWS YARN
  - Distributed XGBoost on Kubernetes
  - Distributed XGBoost with XGBoost4J-Spark
  - Distributed XGBoost with Dask
  - Distributed XGBoost with Ray
  - DART booster
  - Monotonic Constraints
  - Random Forests(TM) in XGBoost



Objective Function: Training Loss + Regularization

With judicious choices for y_i, we may express a variety of tasks, such as regression, classification, and ranking. The task of **training** the model amounts to finding the best parameters θ that best fit the training data x_i and labels y_i. In order to train the model, we need to define the **objective function** to measure how well the model fit the training data.

A salient characteristic of objective functions is that they consist of two parts: **training loss** and **regularization term**:

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

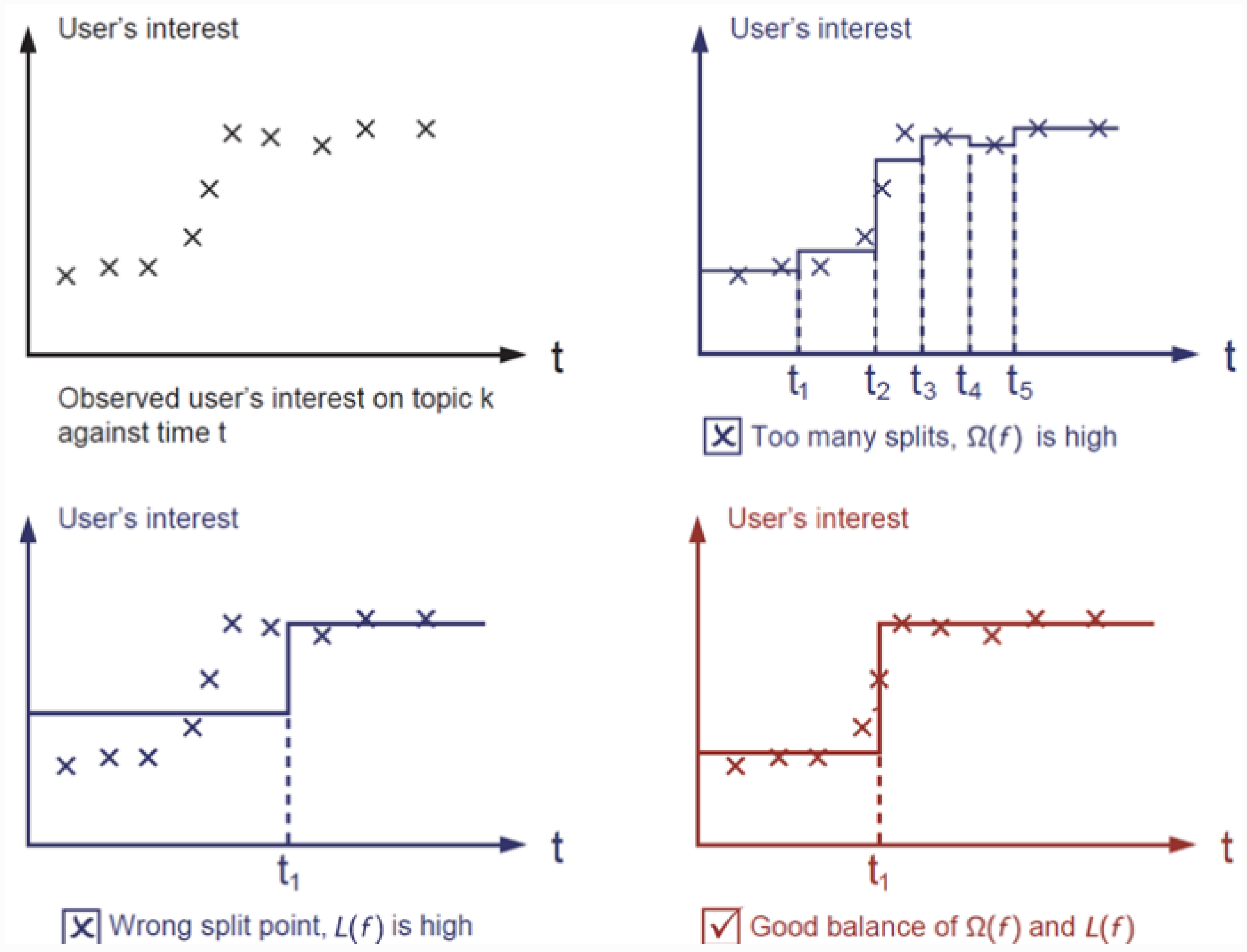
where L is the training loss function, and Ω is the regularization term. The training loss measures how *predictive* our model is with respect to the training data. A common choice of L is the *mean squared error*, which is given by

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

Another commonly used loss function is logistic loss, to be used for logistic regression:

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$$

The **regularization term** is what people usually forget to add. The regularization term controls the complexity of the model, which helps us to avoid overfitting. This sounds a bit abstract, so let us consider the following problem in the following picture. You are asked to *fit* visually a step function given the input data points on the upper left corner of the image. Which solution among the three do you think is the best fit?



Wavelet-based gradient boosting

We again cover the case of regression & classification with the same model / approach

$$\{(x_i, \vec{y}_i)\}_{i\in I}, \ x_i \in \mathbb{R}^n, \vec{y}_i \in \mathbb{R}^L, \ |I_{\text{train}}| = N, \ \text{Loss} = \frac{1}{N}\sum_{i\in I_{\text{train}}} \left\| \tilde{f}(x_i) - \vec{y}_i \right\|^2_{l_2(\mathbb{R}^L)}$$

We construct a sequence of pruned trees, each considered a “weak learner” (e.g. XGBoost). One simpler version of this is to construct trees with limited depth.

At each iteration, current model = weighted voting of the trees.

At each iteration, we ‘boost’ the samples which the current model has the largest error on

One may initialize the model with $\hat{f}_0(x) = \frac{1}{N}\sum_{i=1}^{N} \vec{y}_i, \ \vec{y}_i^{(0)} = \vec{y}_i, \ i \in I_{\text{train}}.$

Wavelet-based gradient boosting

For $k = 1, \ldots, K$

(a) Update residuals $\vec{y}_i^{(k)} = \vec{y}_i - \hat{f}_{k-1}(x_i),\ i \in I_{\text{train}}.$

(b) Choose random validation set (OOB=Out Of Bag) $J_k \subset I_{\text{train}}$

(c) Construct a tree $\mathcal{T}_k$ using $\left(x_i, \vec{y}_i^{(k)}\right),\ i \in I_{\text{train}} \setminus J_k$

(d) Select best M for M-term wavelet approximation $\sum_{m=1}^{M} \psi_{\Omega_m},\ \ \Omega_m \in \mathcal{T}_k$
of k-th validation set $\left(x_i, \vec{y}_i^{(k)}\right),\ i \in J_k$

(e) Update the prediction model with learning rate $\nu > 0$

$$\hat{f}_k = \hat{f}_{k-1} + \nu \sum_{m=1}^{M} \psi_{\Omega_m}$$

Wavelet-based gradient boosting

Datasets details

Dataset	#Samples	#Features
Banana	5299	2
PID	768	8
Heart	270	13
TwoNorm	7400	20

Comparison on datasets with mislabeling (% accuracy)

Dataset name	Noise Level	rAdaBoost	rBoost-Fixed γ	rBoost	GBoost	MBoost	WGB
Banana	0.1	86.87 ± 1.1	87.06 ± 0.9	87.04 ± 0.9	83.91 ± 1.6	78.13 ± 3.4	**87.60 ± 1.6**
	0.3	85.27 ± 3.0	**85.53 ± 2.1**	85.06 ± 2.7	79.38 ± 1.6	75.31 ± 2.5	85.49 ± 1.3
PID	0.1	74.20 ± 2.3	74.37 ± 1.5	**74.80 ± 2.4**	72.60 ± 2.0	75.67 ± 1.9	74.21 ± 6.2
	0.3	72.53 ± 1.9	70.43 ± 2.4	71.43 ± 2.3	69.40 ± 2.9	73.33 ± 2.3	**75.65 ± 7.5**
Heart	0.1	78.40 ± 3.1	79.70 ± 3.5	79.10 ± 4.4	76.40 ± 3.1	77.60 ± 3.5	**80.74 ± 8.2**
	0.3	**78.50 ± 4.0**	77.40 ± 6.5	78.10 ± 4.3	70.00 ± 5.5	75.20 ± 3.7	73.70 ± 11.5
TwoNorm	0.1	95.70 ± 0.8	95.58 ± 0.9	95.59 ± 0.7	90.35 ± 1.0	92.79 ± 0.5	**96.40 ± 0.8**
	0.3	93.33 ± 0.9	93.13 ± 1.3	93.40 ± 1.1	83.94 ± 2.0	91.16 ± 0.9	**94.82 ± 0.7**