Boosting Trees

Boosting

- **Boosting** is an ensemble meta-algorithm for primarily reducing bias, and also variance in supervised **learning**
- "Can a set of weak learners create a single strong learner?"
- A weak learner is defined to be a classifier that is only slightly correlated with the true classification

- **https://en.wikipedia.org/wiki/Boosting_(machine_learning)**

AdaBoost: Summary

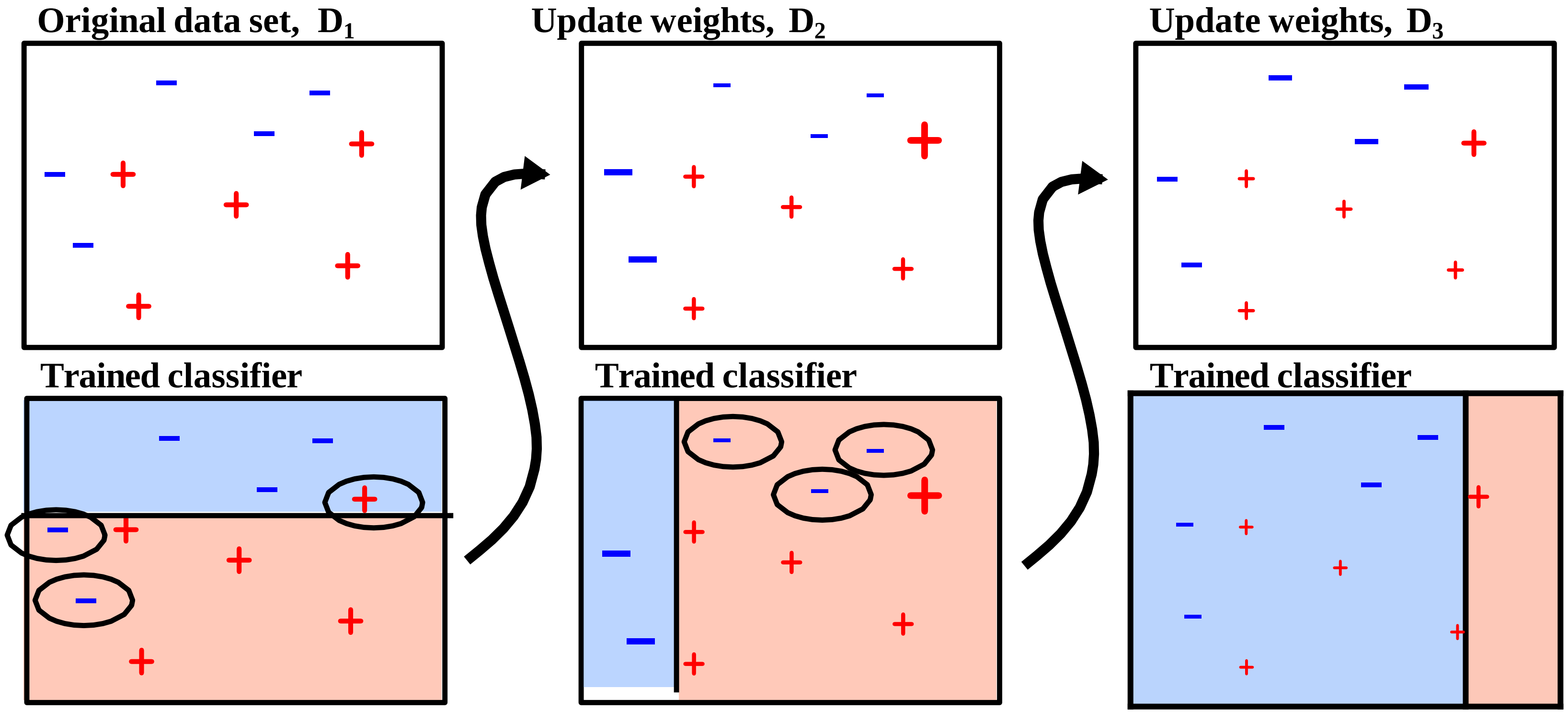
- **Misclassified** samples receive **higher weight.** The higher the weight the **"more attention"** a training sample receives

- Algorithm generates weak classifier by training the next learner **on the mistakes** of the previous one

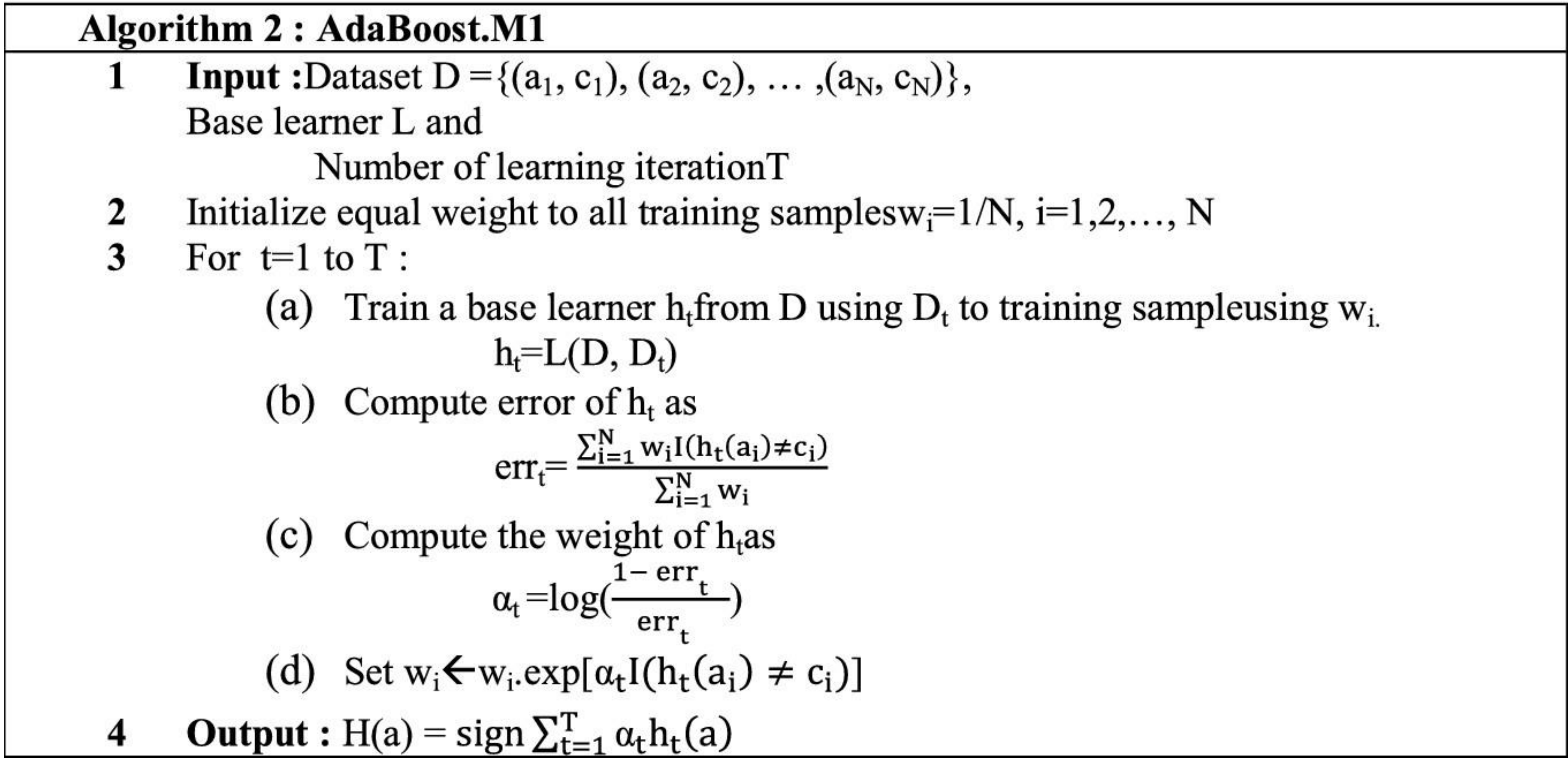
- AdaBoost **minimizes the upper bound** of the training error by properly choosing the optimal weak classifier and voting weight.

Boosting Example

Classes +1 , -1



https://en.wikipedia.org/wiki/AdaBoost



Algorithm 2 : AdaBoost.M1

```
1   Input :Dataset D ={(a_1, c_1), (a_2, c_2), … ,(a_N, c_N)},
    Base learner L and
            Number of learning iterationT
2   Initialize equal weight to all training samples$w_i$=1/N, i=1,2,…, N
3   For  t=1 to T :
        (a)  Train a base learner $h_t$from D using $D_t$ to training sampleusing $w_{i.}$
                 $h_t=L(D, D_t)$
        (b)  Compute error of $h_t$ as
                 $err_t= \frac{\sum_{i=1}^{N} w_i I(h_t(a_i) \neq c_i)}{\sum_{i=1}^{N} w_i}$
        (c)  Compute the weight of $h_t$as
                 $\alpha_t = \log(\frac{1- err_t}{err_t})$
        (d)  Set $w_i \leftarrow w_i.\exp[\alpha_t I(h_t(a_i) \neq c_i)]$
4   Output : $H(a) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(a)$
```

AdaBoost

- Armies of decision stumps
- AdaBoost works on the training phase, seeing how some classifier who failed during the classification should be payed greater attention (increase the weight)

1. Initially, all observations are given equal weights.
2. A model is built on a subset of data.
3. Using this model, predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

Adaboost Example

pain	block	patient	Class	Weight	UW	NW	
C			1	1/7	0.05	0.07	0 ~ 0.07
IC			0	1/7	0.34	0.51	0.07 ~ 0.58
C			1	1/7	0.05	0.07	0,.58 ~ 0.65
C			0	1/7	0.05	0.07	0.65~0.72
C			0	1/7	0.05	0.07	0.73~0.8
C			1	1/7	0.05	0.07	0.8~0.87
C			1	1/7	0.05	0.07	0.87~0.94

Or Performance of a stump

$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

Incorrect

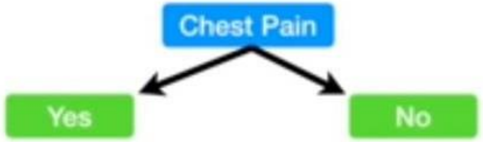
$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

correct

$$\text{New Sample Weight} = \text{sample weight} \times e^{-\text{amount of say}}$$

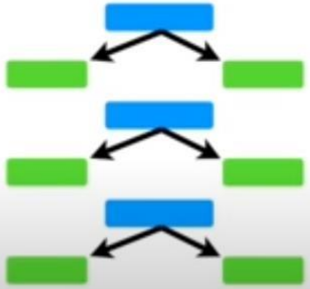
https://towardsdatascience.com/adaboost-for-dummies-breaking-down-the-math-and-its-equations-into-simple-terms-87f439757dcf

https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464

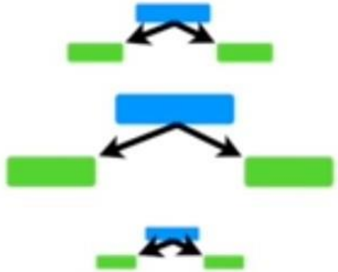


Thus, **Stumps** are technically "weak learners".

1) **AdaBoost** combines a lot of "weak learners" to make classifications. The weak learners are almost aways **stumps**.

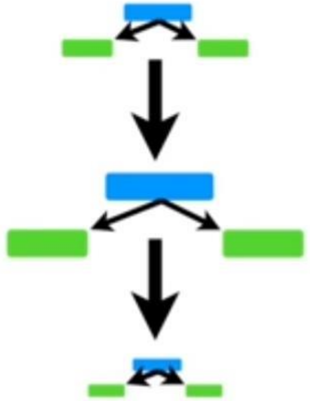


2) Some **stumps** get more say in the classification than others.



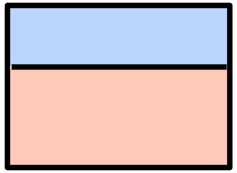
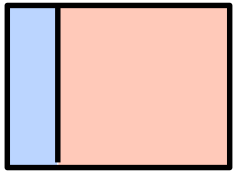
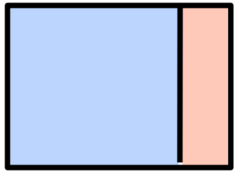
re say in
others.

3) Each **stump** is made by taking the previous **stump's** mistakes into account.



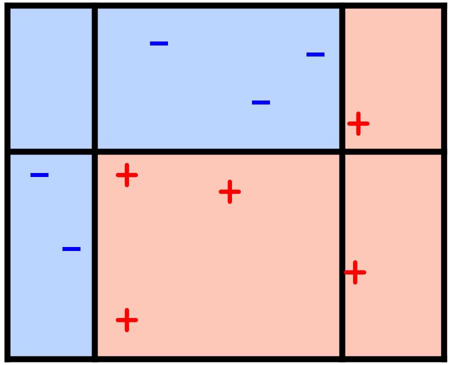
Boosting Example

Weight each classifier and combine them:

.33 *  **+ .57 ***  **+ .42 ***  $\gtrless$ **0**

)

Combined classifier



1-node decision trees
"decision stumps" ***very simple classifiers***

Gradient boosting

Gradient boosting algorithms are some of the most powerful machine learning algorithms for classification and regression

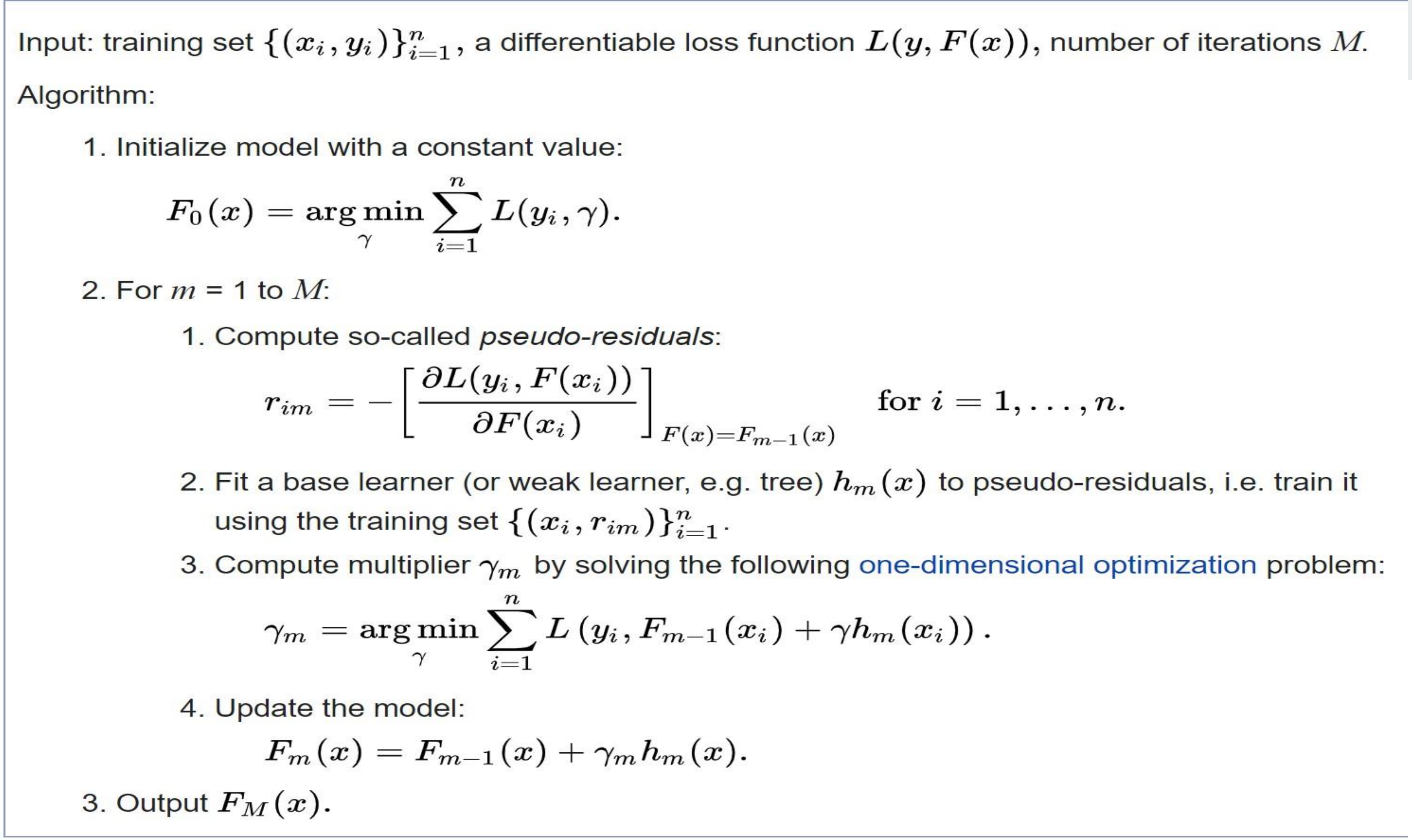
Popular implementations include:

- AdaBoost
- CatBoost (a recently open sourced gradient boosting library from Yandex)
- Spark MLlib also includes its own gradient-boosted tree (GBT) implementation
- **XGBoost** (eXtreme Gradient Boosting) is one of the best gradient-boosted tree implementations currently available

Gradient Boosting

https://en.wikipedia.org/wiki/Gradient_boosting

- A special case of boosting where errors are minimized by gradient descent algorithm, Use loss function, weak learner
- We want the algorithm to generalize.
- Gradient on the other hand is defined only on the training data points.
- So fit the tree T to the negative gradient values by least squares.



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:

    1. Compute so-called *pseudo-residuals*:

    $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

    $$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

    $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Gradient boost Example

pain	area	H Price	Y_hat	r1	F2	
		50K	60K	-10	59	
		70K	60	10	61	
		60K	60	0	60	
		60K	60	0	60	

Loss = $\sum_{i=1}^{n}$ ½ (y - y_hat) **2

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1} L(y_i, \gamma).$$

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

https://www.gormanalysis.com/blog/gradient-boosting-explained/

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4

Tuning the Parameters

- The parameters that can be tuned are
  - The size of constituent trees J.
  - The number of boosting iterations M.
  - Shrinkage (learning rate)

Right-sized trees

- The optimal for one step might not be the optimal for the algorithm
    - Using very large tree as weak learner to fit the residue assumes each tree is the last one in the expansion. Usually degrade performance and increase computation
- Solution : restrict the value of J to be the same for all trees.
- In practice the value of $4 \leq J \leq 8$ is seen to have worked the best.

XGBoosting http://dmlc.cs.washington.edu/data/pdf/XGBoostArxiv.pdf

- **XGBoost**: Extreme Gradient Boosting uses a gradient boosting framework to minimize the loss when adding new models.
- Gradient Booting with parallelization, create DT parallelization,
- https://en.wikipedia.org/wiki/XGBoost

```
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

XGBoost

- The original XGBoost paper can be found
- https://arxiv.org/pdf/1603.02754.pdf

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

prediction of the i-th instance at the t-th iteration, we will need to add f_t to minimize the following objective.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

This means we greedily add the f_t that most improves our model according to Eq. (2). Second-order approximation can be used to quickly optimize the objective in the general setting [12].

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ are first and second order gradient statistics on the loss function. We can remove the constant terms to obtain the following simplified objective at step t.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \qquad (3)$$

Math Explained: https://towardsdatascience.com/xgboost-mathematics-explained-58262530904a

Example: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

XGBoost

- XGBoost has become the dominant machine learning algorithm for classification and regression.
- Designed for efficiency and scalability, its parallel tree boosting capabilities make it significantly faster than other tree-based ensemble algorithms. Due to its high accuracy, XGBoost has gained popularity by winning several machine learning competitions.
- In 2015, 17 out of the 29 winning solutions on Kaggle used XGBoost. All the top 10 solutions at the KDD Cup 2015xxiv used XGBoost.

Design principles of XGBoost

- Combining weak learners into a strong learner
- New models are added to correct the errors made by existing models

Difference between other gradient-boosted trees and XGBoost:

- XGBoost builds trees in parallel - due to which it is faster

- Other gradient-boosted trees are built sequentially – slowly learning from data to improve its prediction in succeeding iteration

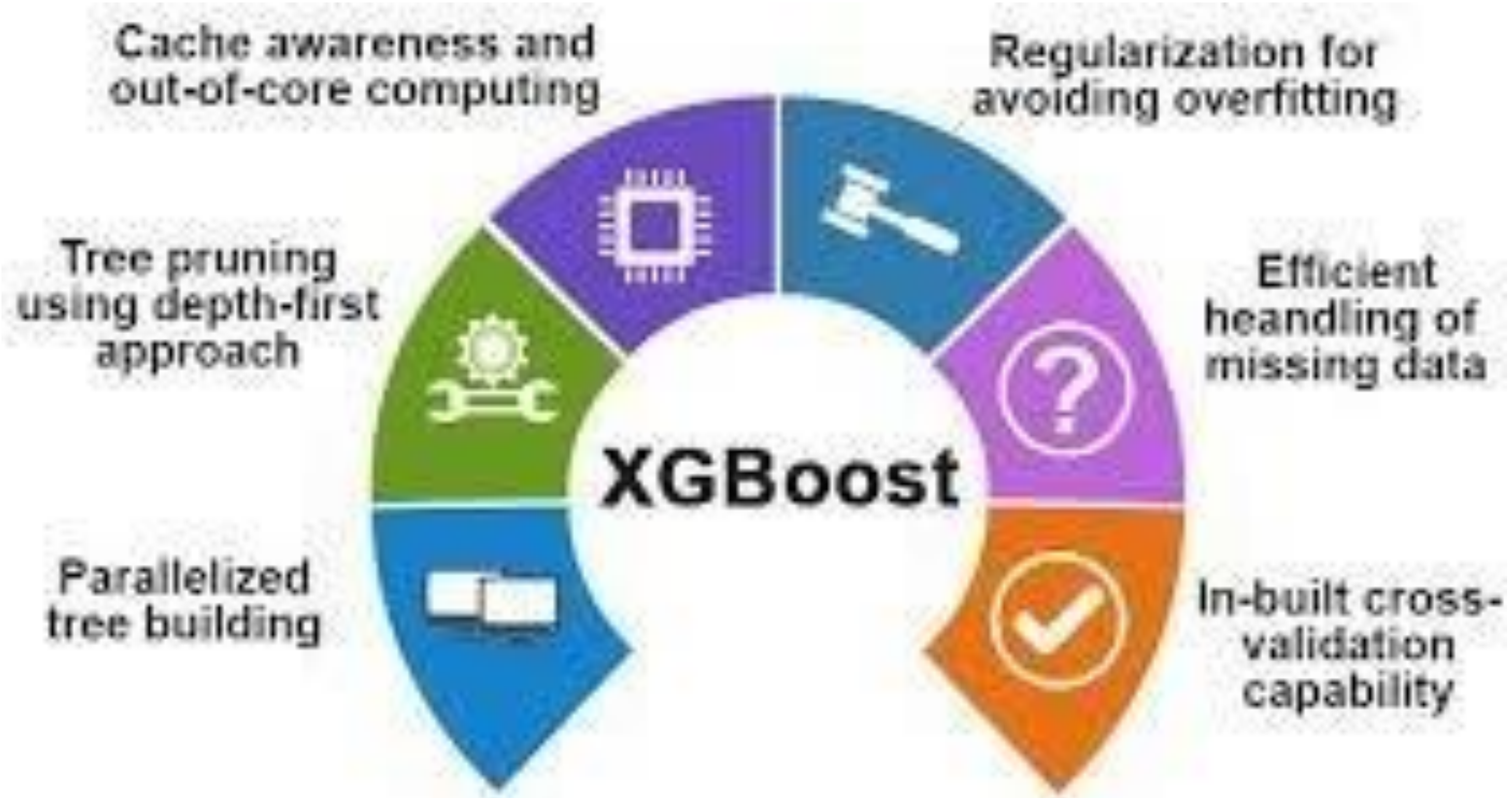
Advantages of parallel boosting

XGBoost produces **better prediction performance** by **controlling model complexity** and **reducing overfitting** through its built-in regularization.

It uses an approximate algorithm to find split points when finding the best split points for a continuous feature.

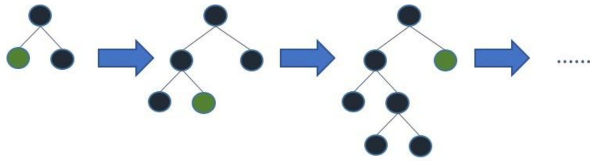
XGBoost Hyperparameters

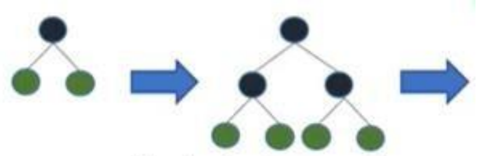
- Booster (gbtree, gblinear)
- Objective (softmax, softprob)
- Eta (learning rate)
- Max_depth (depth of tree)
- Etc...
- GridSearchCV (Grid Search with Cross-Validation) is a brute force on finding the best hyperparameters for a specific dataset and model


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient heandling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

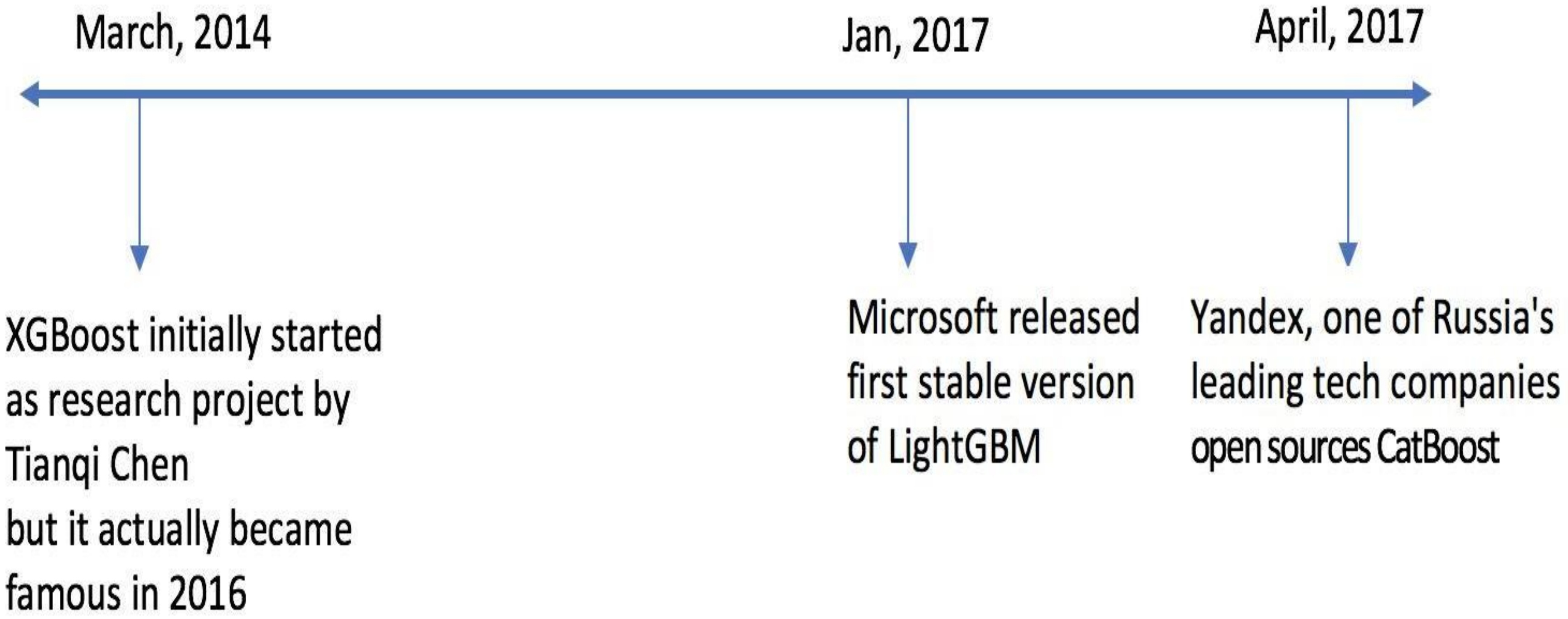
Light GBM (https://en.wikipedia.org/wiki/LightGBM)

- https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d
- **Faster training speed and higher efficiency**:
- **Lower memory usage:** Replaces continuous values to discrete bins which result in lower memory usage.
- **Better accuracy than other boosting algorithm:** .
- **Compatibility with Large Datasets:** It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.


Leaf-wise tree growth


Level-wise tree growth

Time-line



March, 2014

XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016

Jan, 2017

Microsoft released first stable version of LightGBM

April, 2017

Yandex, one of Russia's leading tech companies open sources CatBoost