CS6375: Machine Learning

Gautam Kunapuli

Ensemble Methods: Boosting

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Gradient Boosting: State of the Art

Highly Bothersome, Yet Utterly Unavoidable Question: Which machine learning algorithm should I use for this data set?

An extensive study in **2014 by Fernández-Delgado et al**[1] evaluated 179 classifiers from 17 families on 121 **[UCI benchmark]** data sets

- "… classifiers **most likely to be the bests are the random forest (RF)**… achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets."
- "… **difference is not statistically significant with the second best, the SVM with Gaussian kernel**… which achieves 92.3% of the maximum accuracy."

A newer study in **2018 by Olson et al** [2] performed "… a thorough analysis of 13 state-of-the-art, commonly used machine learning algorithms on a set of 165 publicly available **[bioinformatics]** classification problems…"

- previous study did not consider gradient boosting
- in both studies, it is worth noting that **no one ML algorithm performs best across all datasets**

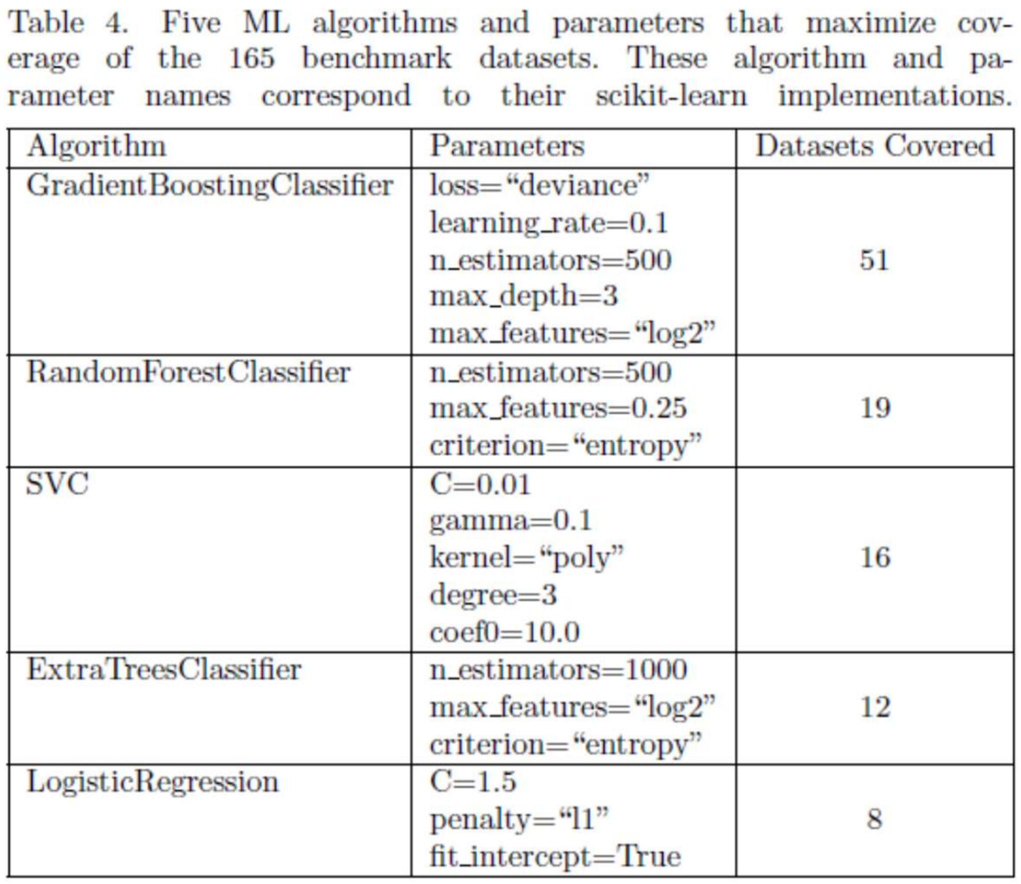


Table 4. Five ML algorithms and parameters that maximize coverage of the 165 benchmark datasets. These algorithm and parameter names correspond to their scikit-learn implementations.

Algorithm	Parameters	Datasets Covered

GradientBoostingClassifier	loss="deviance"
	learning_rate=0.1
	n_estimators=500
	max_depth=3
	max_features="log2"
		51

RandomForestClassifier	n_estimators=500
	max_features=0.25
	criterion="entropy"
		19

SVC	C=0.01
	gamma=0.1
	kernel="poly"
	degree=3
	coef0=10.0
		16

ExtraTreesClassifier	n_estimators=1000
	max_features="log2"
	criterion="entropy"
		12

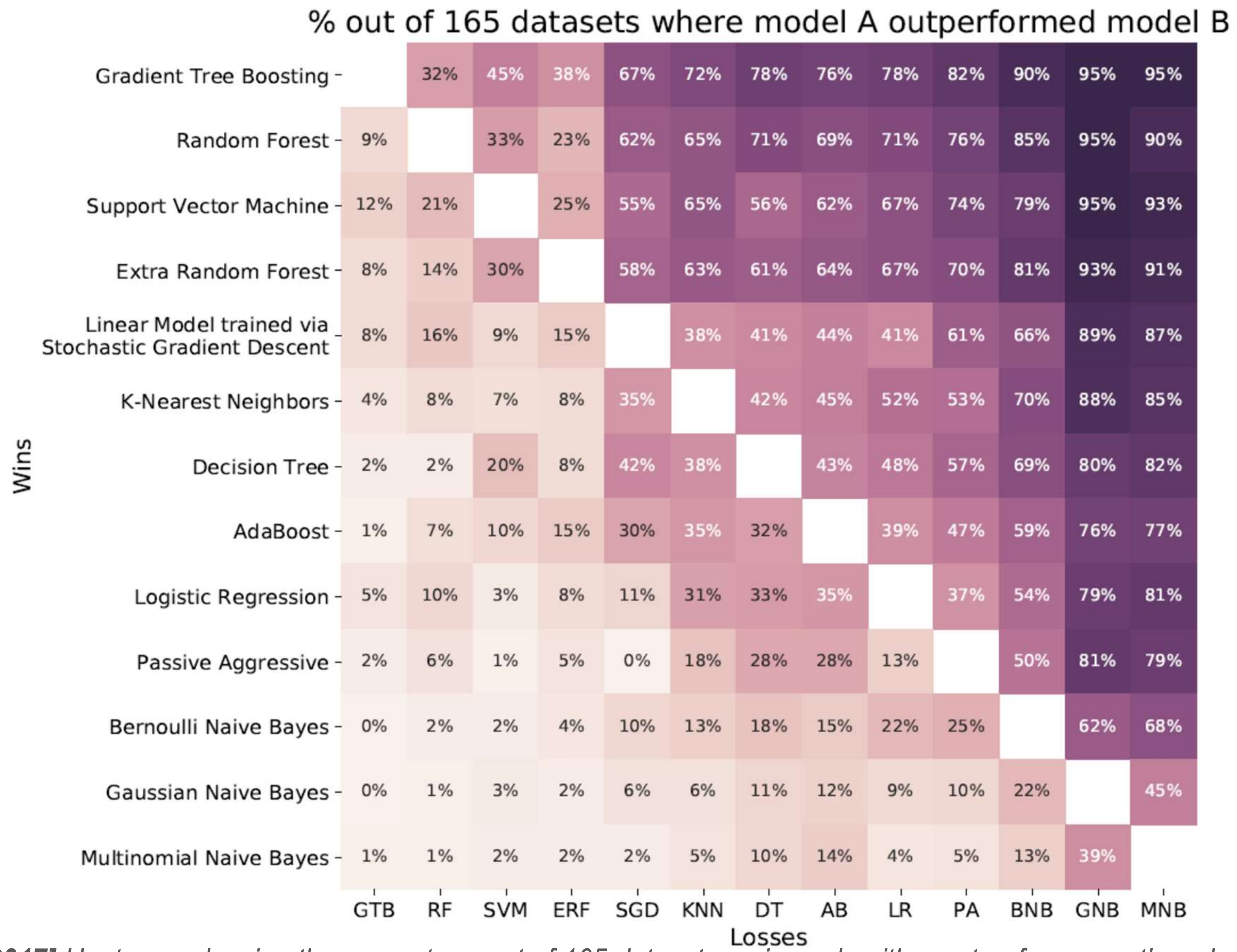
LogisticRegression	C=1.5
	penalty="l1"
	fit_intercept=True
		8

*Results from **[Olson et al, 2018]***

[1] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. **Do we need hundreds of classifiers to solve real world classification problems?** *J. Mach. Learn. Res.* 15, 1 (January 2014), 3133-3181. http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf

[2] Olson RS, La Cava W, Mustahsan Z, Varik A, Moore JH. **Data-driven advice for applying machine learning to bioinformatics problems**. *Pacific Symposium on Biocomputing*. 2018;23:192-203. https://arxiv.org/pdf/1708.05070.pdf

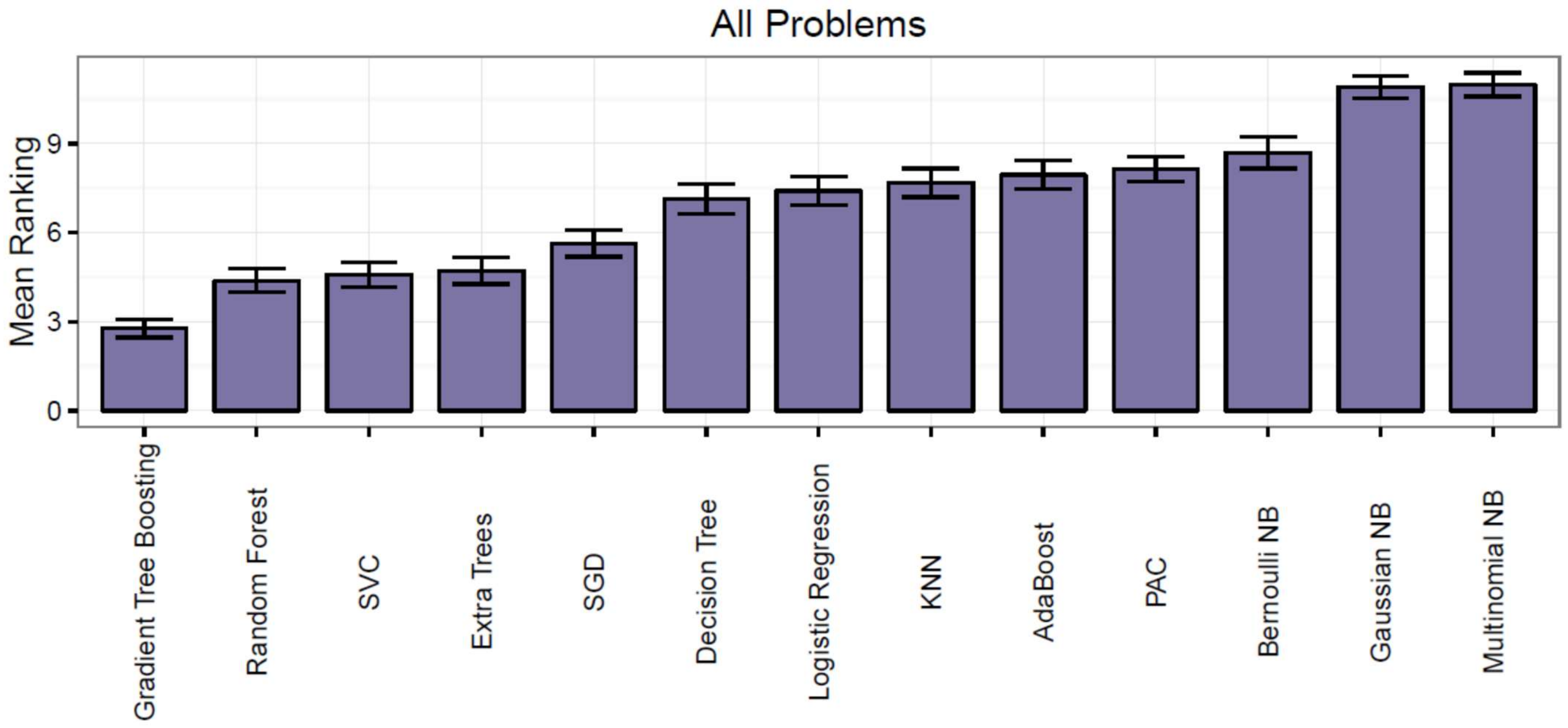
Gradient Boosting: State of the Art



	GTB	RF	SVM	ERF	SGD	KNN	DT	AB	LR	PA	BNB	GNB	MNB
Gradient Tree Boosting		32%	45%	38%	67%	72%	78%	76%	78%	82%	90%	95%	95%
Random Forest	9%		33%	23%	62%	65%	71%	69%	71%	76%	85%	95%	90%
Support Vector Machine	12%	21%		25%	55%	65%	56%	62%	67%	74%	79%	95%	93%
Extra Random Forest	8%	14%	30%		58%	63%	61%	64%	67%	70%	81%	93%	91%
Linear Model trained via Stochastic Gradient Descent	8%	16%	9%	15%		38%	41%	44%	41%	61%	66%	89%	87%
K-Nearest Neighbors	4%	8%	7%	8%	35%		42%	45%	52%	53%	70%	88%	85%
Decision Tree	2%	2%	20%	8%	42%	38%		43%	48%	57%	69%	80%	82%
AdaBoost	1%	7%	10%	15%	30%	35%	32%		39%	47%	59%	76%	77%
Logistic Regression	5%	10%	3%	8%	11%	31%	33%	35%		37%	54%	79%	81%
Passive Aggressive	2%	6%	1%	5%	0%	18%	28%	28%	13%		50%	81%	79%
Bernoulli Naive Bayes	0%	2%	2%	4%	10%	13%	18%	15%	22%	25%		62%	68%
Gaussian Naive Bayes	0%	1%	3%	2%	6%	6%	11%	12%	9%	10%	22%		45%
Multinomial Naive Bayes	1%	1%	2%	2%	2%	5%	10%	14%	4%	5%	13%	39%	

***[Olson et al, 2017]** Heat map showing the percentage out of 165 datasets a given algorithm outperforms another algorithm in terms of best accuracy on a problem. The algorithms are ordered from top to bottom based on their overall performance on all problems. Two algorithms are considered to have the same performance on a problem if they achieved an accuracy within 1% of each other.*

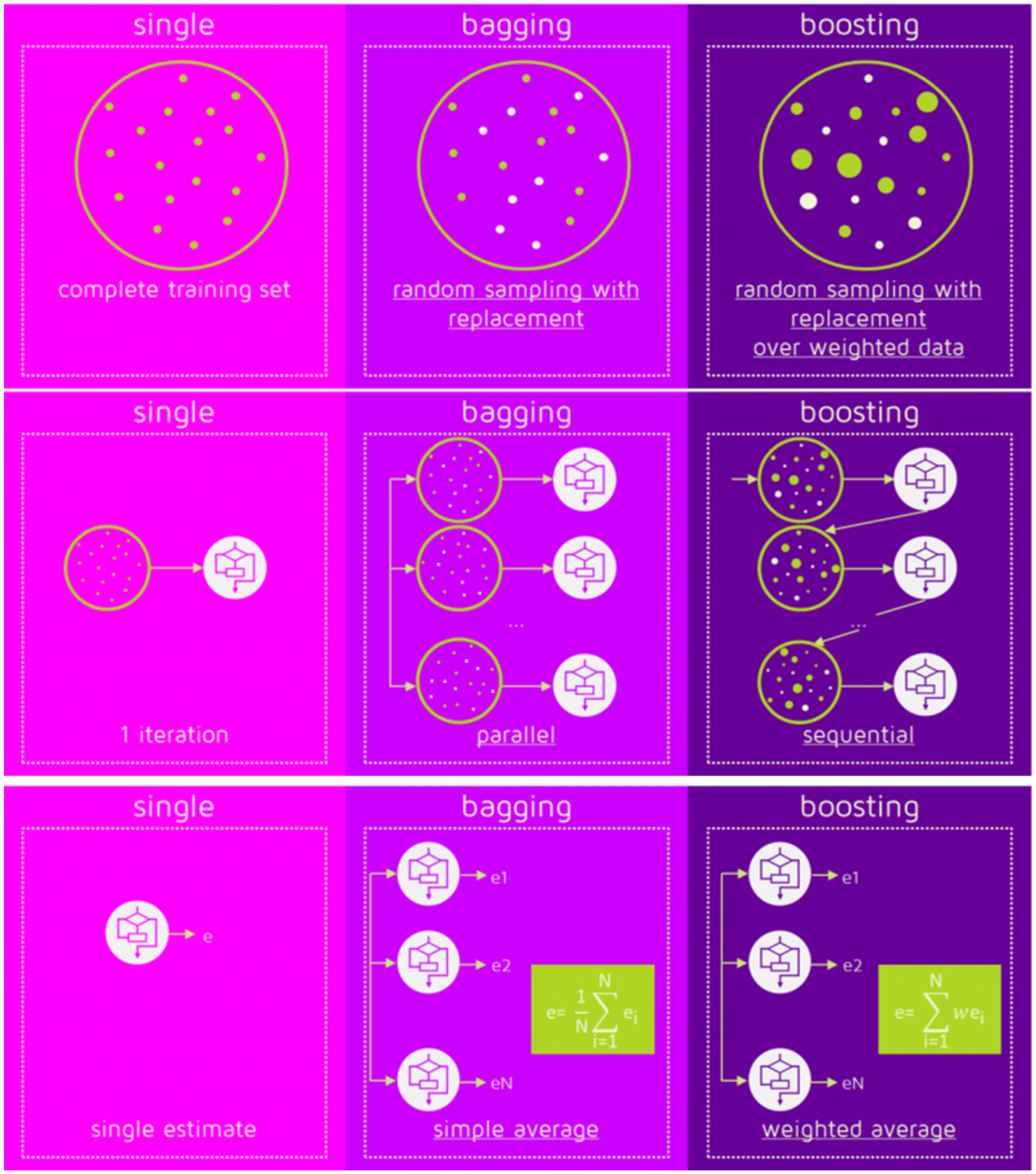
Gradient Boosting: State of the Art



***[Olson et al, 2017]** Average ranking of the ML algorithms over all datasets. Error bars indicate the 95% confidence interval.*

Highly Bothersome, Yet Utterly Unavoidable Question: Which machine learning algorithm should I use for this data set?

A Review of Bagging and Boosting



Bagging

- Uses **bootstrap sampling** to learn many (strong) models, whose predictions can be **aggregated**
- bagging reduces **variance** (and in practice, maybe increases bias slightly)
- bagging learns with **bootstrap samples** of the **same size as the original data set**
  - with decision trees, typically learns full trees
  - computational complexity is higher
- each model is learned independently of other models
  - insight from one model does not influence the learning of the next model

Boosting

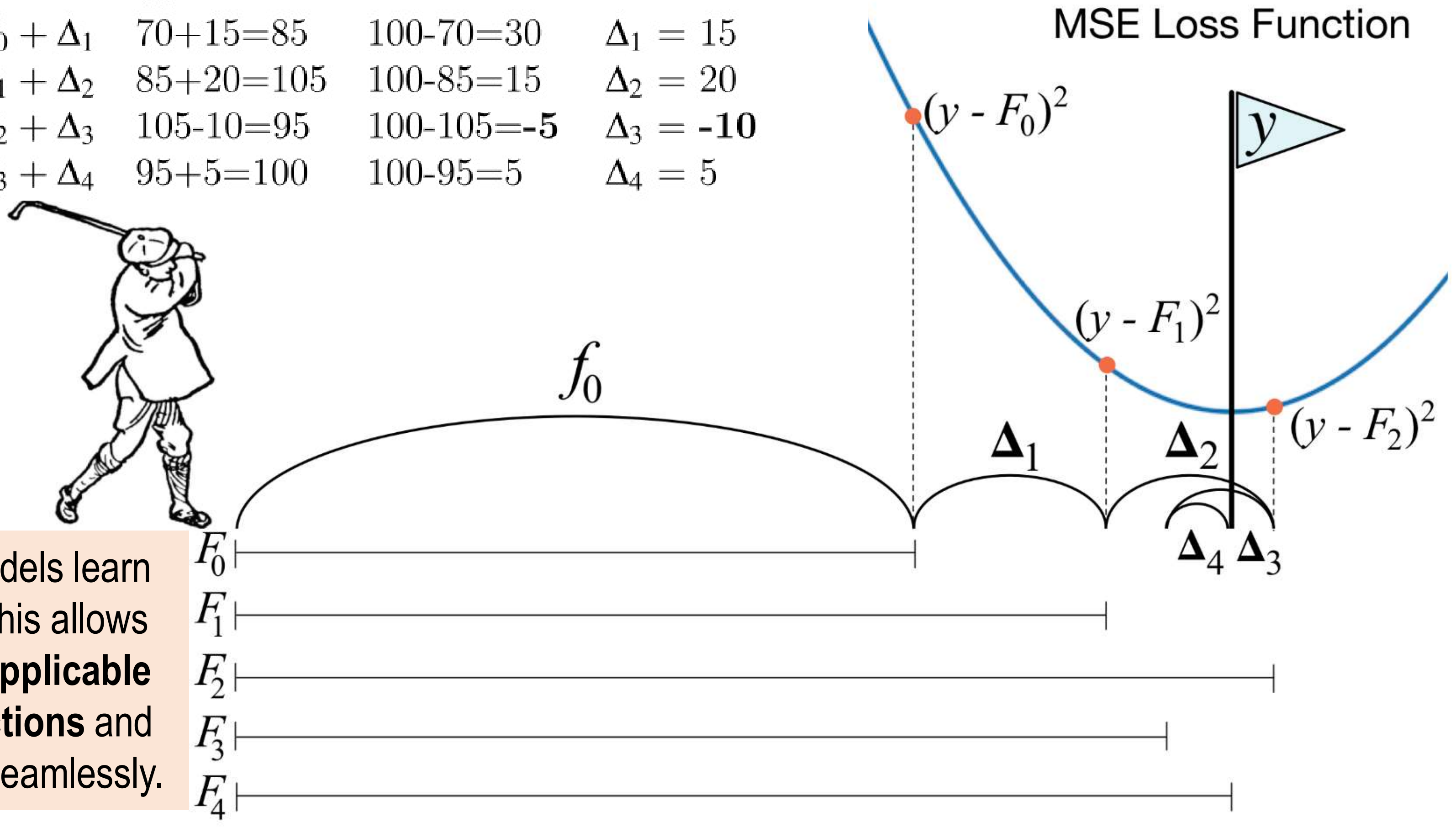
- uses weak learners that have high bias
  - decision stumps (decision trees with depth 1)
- boosting reduces both **bias** and **variance**
- **iterative algorithm** that increases weights on hard examples
  - insight from previous iterations guides learning

Images from https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Gradient Boosting: Intuition with Regression

- In **adaptive boosting** (AdaBoost), shortcomings are identified by **high-weight data points**
- in **gradient boosting**, shortcomings are identified by **gradients**
- both high-weight data points and gradients tell us how to improve our model
- **gradient boosting = gradient descent + boosting**

Stage m	Boosted Model	Model Output $\hat{y}$	Train Δ_m on $y - F_{m-1}$	Noisy Prediction Δ_m
0	F_0	70		
1	$F_1 = F_0 + \Delta_1$	70+15=85	100-70=30	$\Delta_1 = 15$
2	$F_2 = F_1 + \Delta_2$	85+20=105	100-85=15	$\Delta_2 = 20$
3	$F_3 = F_2 + \Delta_3$	105-10=95	100-105=**-5**	$\Delta_3 =$ **-10**
4	$F_4 = F_3 + \Delta_4$	95+5=100	100-95=5	$\Delta_4 = 5$



Main intuition: weak models learn **gradient information**! This allows gradient boosting to **be applicable to a variety of loss functions** and extend to problem types seamlessly.

Figure from http://explained.ai/gradient-boosting/index.html

Gradient Boosting: Intuition with Regression

At the m-th iteration, we wish to improve the previous model's prediction $\hat{y} = F_{m-1}(\boldsymbol{x})$ such that for all data points

$$F_{m-1}(\boldsymbol{x}_1) + \boldsymbol{\Delta}_m(\boldsymbol{x}_1) = y_1$$
$$\dots$$
$$F_{m-1}(\boldsymbol{x}_i) + \boldsymbol{\Delta}_m(\boldsymbol{x}_i) = y_i$$
$$\dots$$
$$F_{m-1}(\boldsymbol{x}_N) + \boldsymbol{\Delta}_m(\boldsymbol{x}_N) = y_N$$

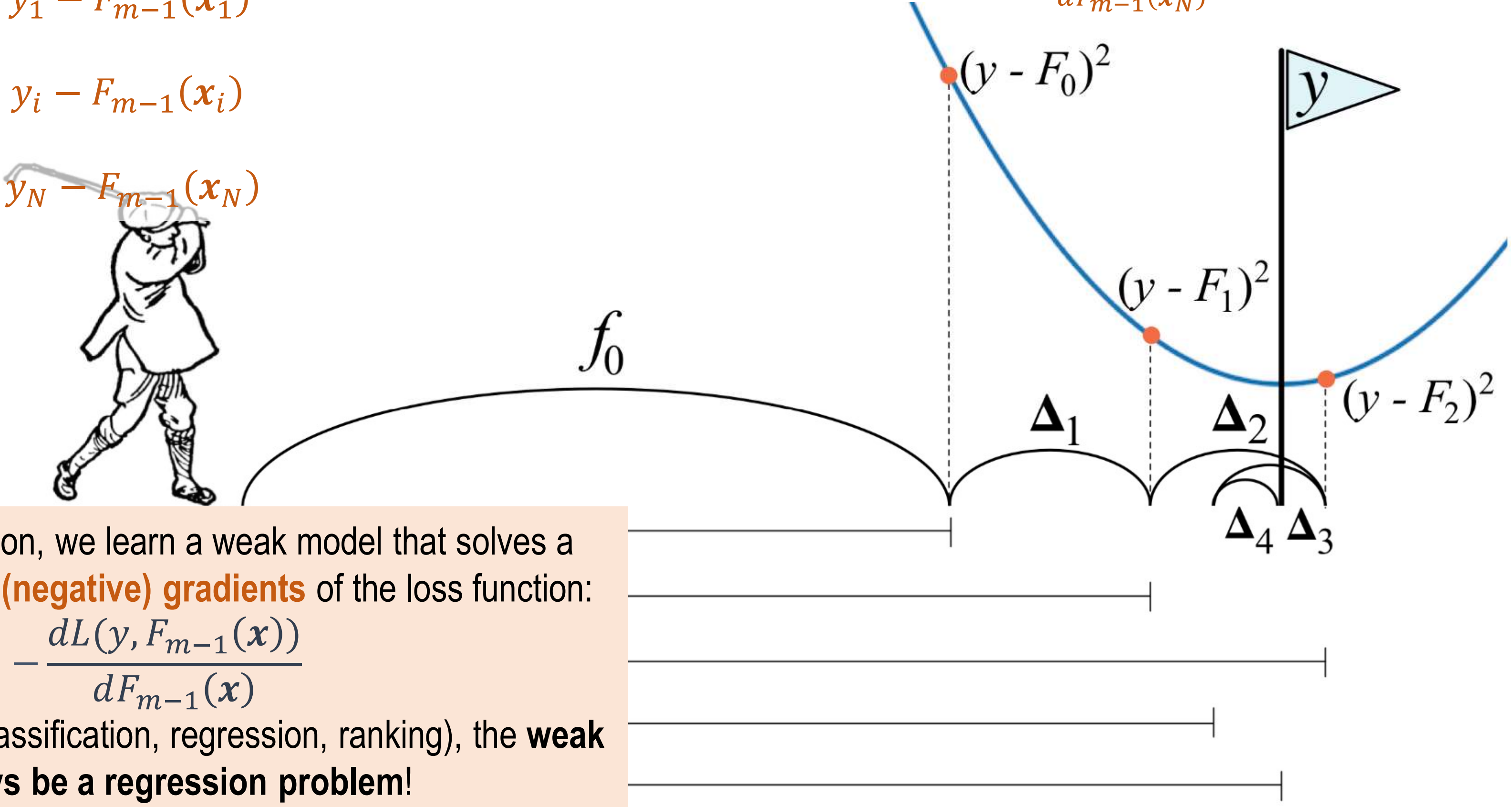
Rewriting in terms of the residuals, $\boldsymbol{\Delta}_m(\boldsymbol{x}) = y - F_{m-1}(\boldsymbol{x})$

$$\boldsymbol{\Delta}_m(\boldsymbol{x}_1) = y_1 - F_{m-1}(\boldsymbol{x}_1)$$
$$\dots$$
$$\boldsymbol{\Delta}_m(\boldsymbol{x}_i) = y_i - F_{m-1}(\boldsymbol{x}_i)$$
$$\dots$$
$$\boldsymbol{\Delta}_m(\boldsymbol{x}_N) = y_N - F_{m-1}(\boldsymbol{x}_N)$$

loss function: $L(y, F_{m-1}(\boldsymbol{x})) = \frac{1}{2}\left(y - F_{m-1}(\boldsymbol{x})\right)^2$

gradient of the loss function: $\frac{dL(y, F_{m-1}(\boldsymbol{x}))}{dF_{m-1}(\boldsymbol{x})} = -(y - F_{m-1}(\boldsymbol{x}))$

$$\boldsymbol{\Delta}_m(\boldsymbol{x}_1) = -\frac{dL(y, F_{m-1}(\boldsymbol{x}_1))}{dF_{m-1}(\boldsymbol{x}_1)}$$
$$\dots$$
$$\boldsymbol{\Delta}_m(\boldsymbol{x}_i) = -\frac{dL(y, F_{m-1}(\boldsymbol{x}_i))}{dF_{m-1}(\boldsymbol{x}_i)}$$
$$\dots$$
$$\boldsymbol{\Delta}_m(\boldsymbol{x}_N) = -\frac{dL(y, F_{m-1}(\boldsymbol{x}_N))}{dF_{m-1}(\boldsymbol{x}_N)}$$



Main intuition: At each iteration, we learn a weak model that solves a **regression problem** to **fit the (negative) gradients** of the loss function:

$$\boldsymbol{\Delta}_m(\boldsymbol{x}) = -\frac{dL(y, F_{m-1}(\boldsymbol{x}))}{dF_{m-1}(\boldsymbol{x})}$$

Irrespective of the main task (classification, regression, ranking), the **weak learner will always be a regression problem**!

Figure from http://explained.ai/gradient-boosting/index.html

(Functional) Gradient Boosting

For any loss function, we need to solve a **regression problem to fit the negative gradients**

$$\boldsymbol{\Delta}_m(\boldsymbol{x}) = -\frac{dL(y, F_{m-1}(\boldsymbol{x}))}{dF_{m-1}(\boldsymbol{x})}$$

squared error: $L(y, F_{m-1}(\boldsymbol{x})) = \frac{1}{2}\left(y - F_{m-1}(\boldsymbol{x})\right)^2$

$$\boldsymbol{\Delta}_m(\boldsymbol{x}_i) = y_i - F_{m-1}(\boldsymbol{x}_i)$$

hinge loss: $L(y, F_{m-1}(\boldsymbol{x})) = \max(0, 1 - yF_{m-1}(\boldsymbol{x}))$

$$\boldsymbol{\Delta}_m(\boldsymbol{x}_i) = \text{step}\left(1 - yF_{m-1}(\boldsymbol{x}_i)\right) \cdot y_i$$

absolute error: $L(y, F_{m-1}(\boldsymbol{x})) = |y - F_{m-1}(\boldsymbol{x})|$

$$\boldsymbol{\Delta}_m(\boldsymbol{x}_i) = \text{sign}(y_i - F_{m-1}(\boldsymbol{x}_i))$$

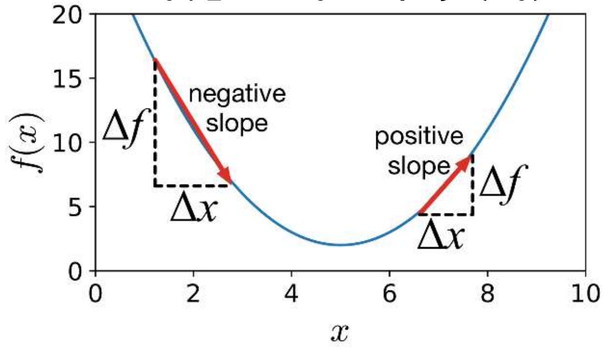
logistic loss: $L(y, F_{m-1}(\boldsymbol{x})) = \log(1 + \exp(-yF_{m-1}(\boldsymbol{x})))$

$$\boldsymbol{\Delta}_m(\boldsymbol{x}_i) = \left(y_i - \frac{1}{1 + \exp(-yF_{m-1}(\boldsymbol{x}_i))}\right) = y_i - P(y = 1|\boldsymbol{x}_i)$$

We take **the gradient with respect to the function** $F_{m-1}(\boldsymbol{x})$; method is also called **functional gradient boosting**. What does this mean?

- Always remember that the prediction is $\hat{y}_{m-1} = F_{m-1}(\boldsymbol{x})$; so gradient descent in function space is the same as **gradient descent in prediction space**!
- the gradient can be written as $-\frac{dL(y, \hat{y}_{m-1})}{d\hat{y}}$, which is just the result of obtaining a prediction $\hat{y}_{m-1}$ using a function $F_{m-1}(\boldsymbol{x})$

Gradient descent: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta\nabla f(\boldsymbol{x}_t)$



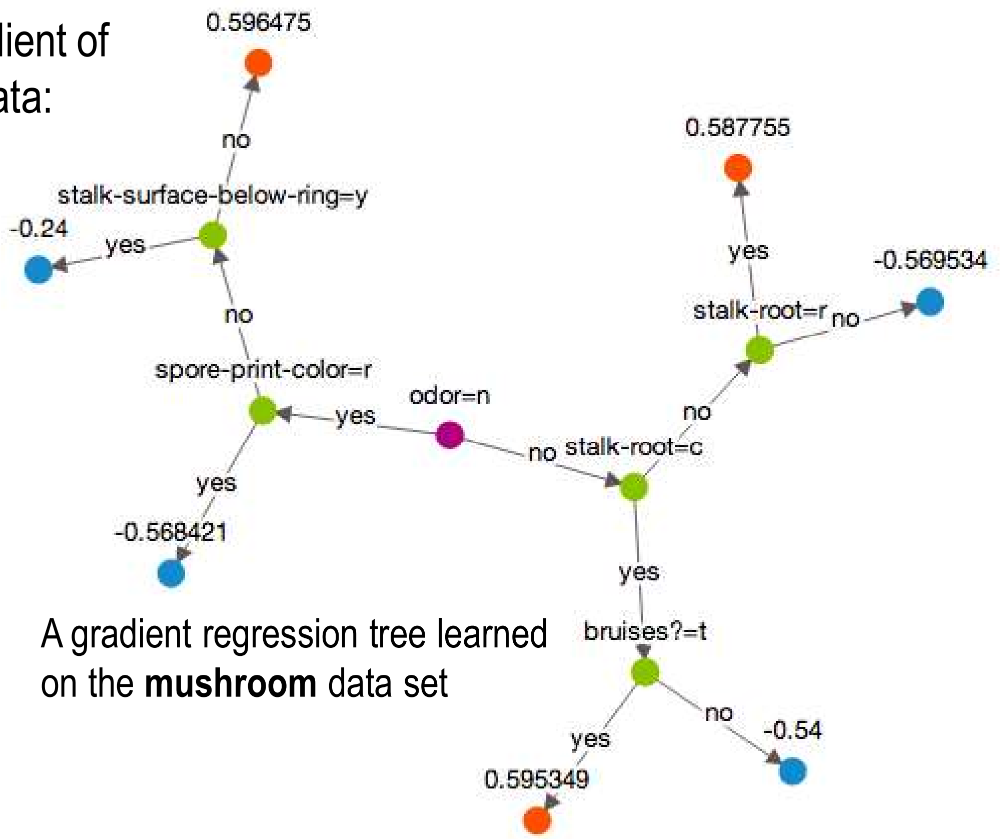
Gradient boosting: $\hat{y}_m = \hat{y}_{m-1} - \eta\nabla L(y, \hat{y}_{m-1})$

Weak Learners: Regression Trees

At iteration m, we must **fit a function** to the (negative) gradient of the loss, which results in the **regression problem** on the data:

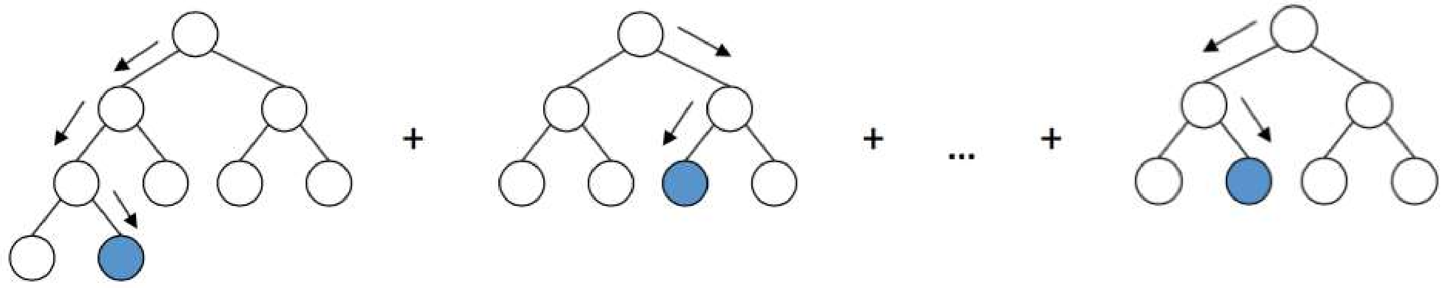
$$\left(\boldsymbol{x}_i, -\frac{dL(y_i, F_{m-1}(\boldsymbol{x}_i))}{dF_{m-1}(\boldsymbol{x}_i)}\right)_{i=1}^{N}$$

Learn gradients as regression trees!

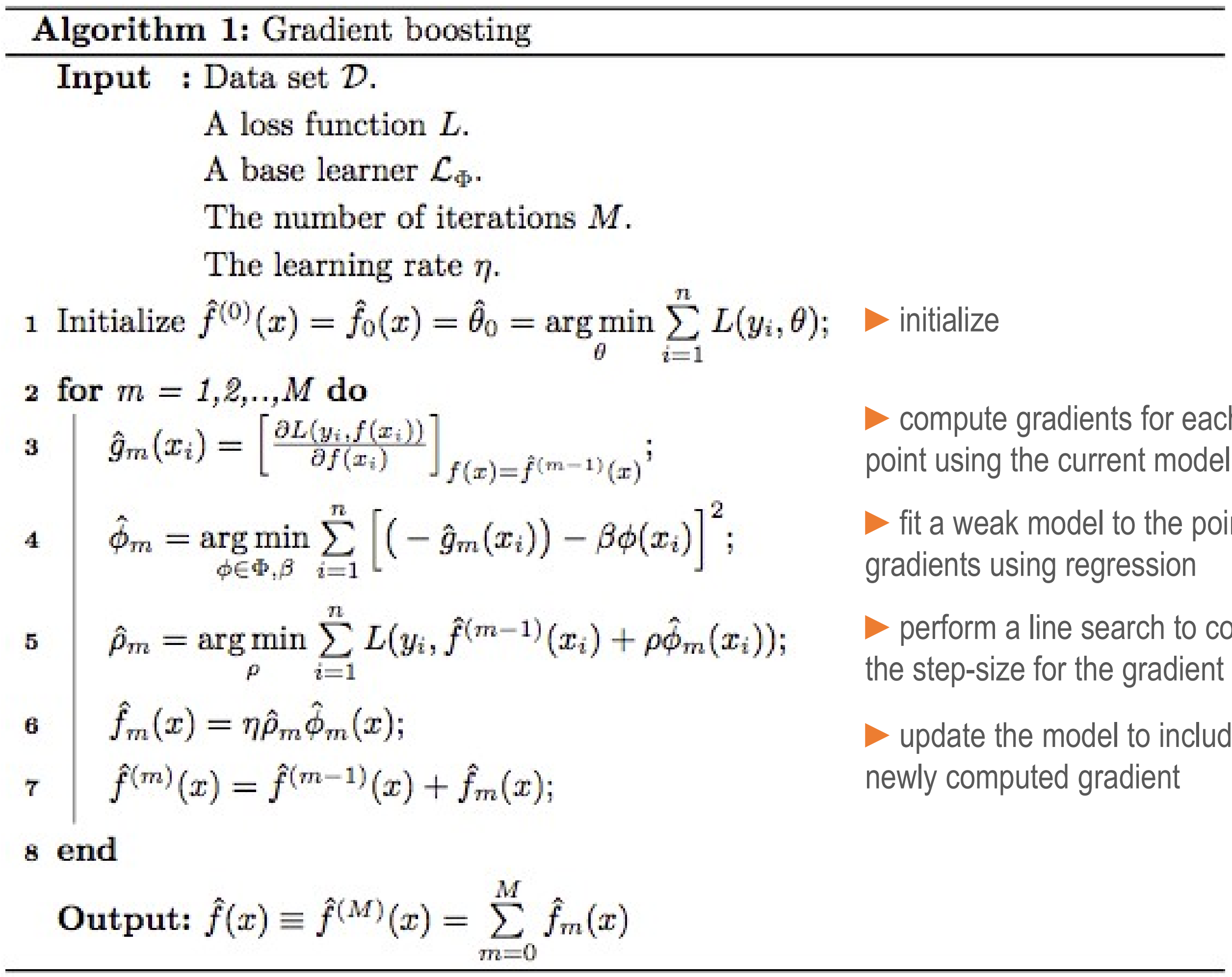


A gradient regression tree learned on the **mushroom** data set

The final model will be a **weighted sum of regression trees**



Gradient Boosting: Full Algorithm



```
Algorithm 1: Gradient boosting
Input  : Data set $\mathcal{D}$.
         A loss function $L$.
         A base learner $\mathcal{L}_\Phi$.
         The number of iterations $M$.
         The learning rate $\eta$.
1 Initialize $\hat{f}^{(0)}(x) = \hat{f}_0(x) = \hat{\theta}_0 = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta)$;
2 for $m$ = 1,2,..,$M$ do
3     $\hat{g}_m(x_i) = \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x) = \hat{f}^{(m-1)}(x)}$;
4     $\hat{\phi}_m = \arg\min_{\phi \in \Phi, \beta} \sum_{i=1}^{n} \left[ \left( -\hat{g}_m(x_i) \right) - \beta \phi(x_i) \right]^2$;
5     $\hat{\rho}_m = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \rho \hat{\phi}_m(x_i))$;
6     $\hat{f}_m(x) = \eta \hat{\rho}_m \hat{\phi}_m(x)$;
7     $\hat{f}^{(m)}(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x)$;
8 end
Output: $\hat{f}(x) \equiv \hat{f}^{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$
```

▶ initialize

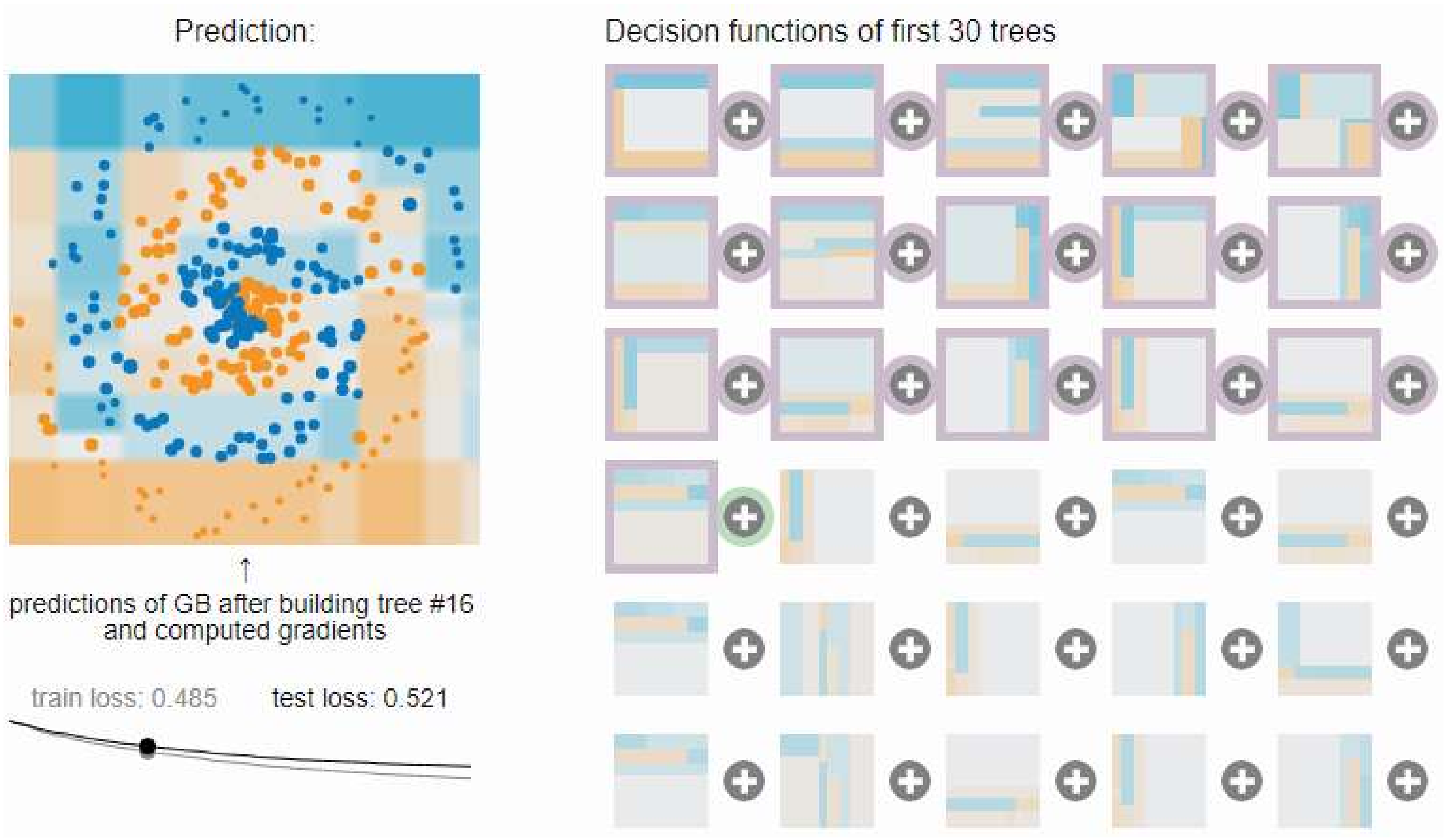
▶ compute gradients for each data point using the current model

▶ fit a weak model to the pointwise gradients using regression

▶ perform a line search to compute the step-size for the gradient

▶ update the model to include the newly computed gradient

Visualizing Gradient Boosting


Prediction:
Decision functions of first 30 trees
predictions of GB after building tree #16
and computed gradients
train loss: 0.485
test loss: 0.521

An excellent interactive playground for gradient boosting can be found here:
http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html

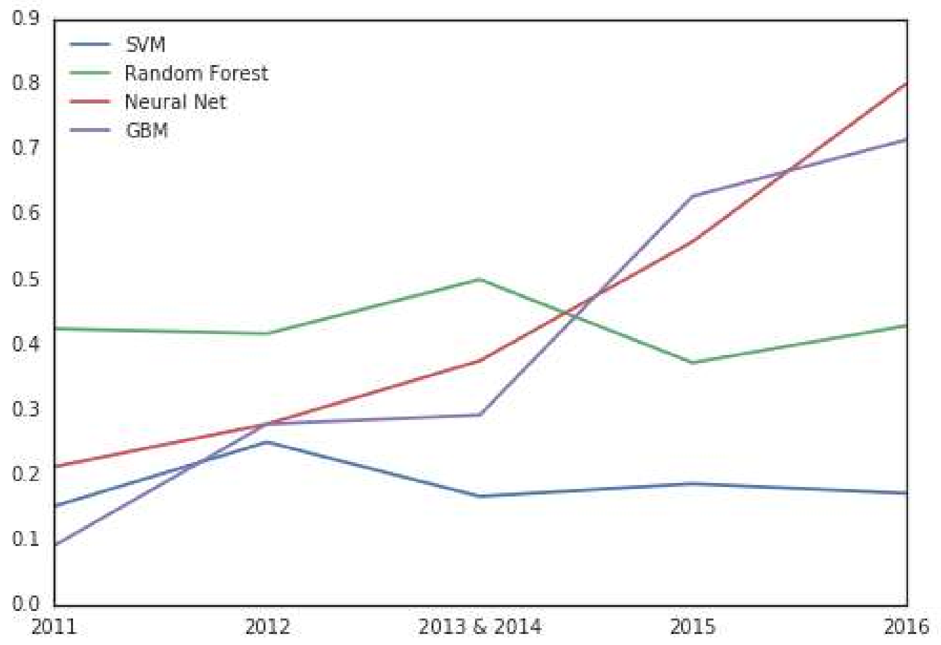
XGBoost: Extreme Gradient Boosting

XGBoost "wins every Kaggle competition"

It extends Gradient Boosting with:

- **Regularization**
  - tree complexity is penalized
  - gradients computed for regularized loss functions
- Proportional **shrinking** of leaf nodes
  - adjusting the **learning rate** slows down learning to prevent overfitting
- **Newton Boosting**
  - uses **second-order derivative** information to speed up convergence
- **Randomization**
  - reduces correlation between trees

Highly **efficient implementations** available for many different programming languages and scientific computing platforms.



Most popular methods mentioned in winners posts are neural networks, SVMs, random forest and GBM. By 2017, over half the winning algorithms on Kaggle were Gradient Boosting and its variants.