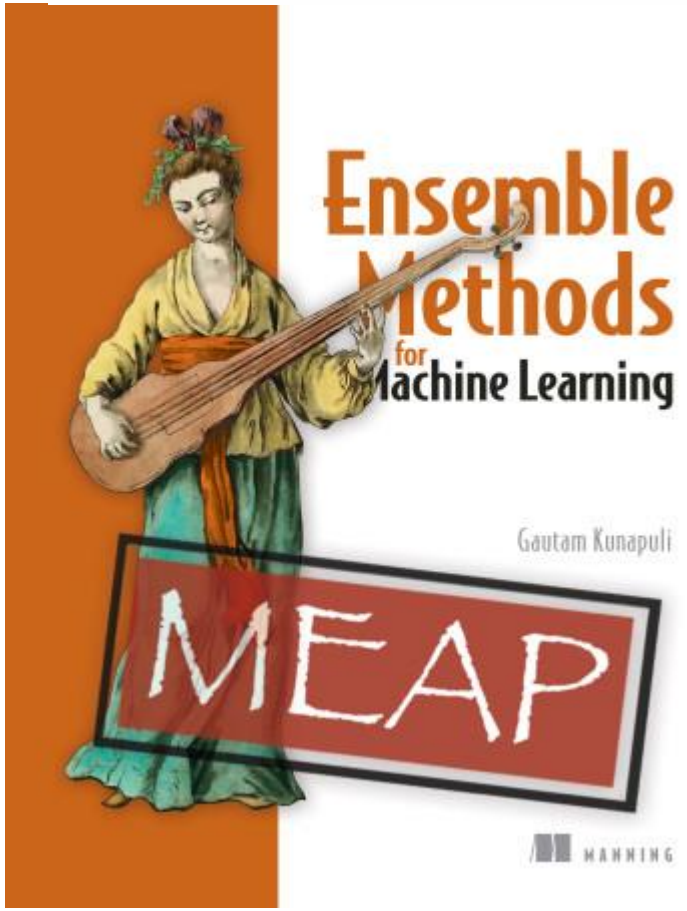
Twitch Live #1

Gradient Boosting From Scratch

Gautam Kunapuli



These Slides (PDF)
https://github.com/gkunapuli/ensemble-methods-notebooks/blob/master/other-materials/GradientBoostingFromScratch.pdf

Manning page for *Ensemble Methods for Machine Learning* (in MEAP)
https://www.manning.com/books/ensemble-methods-for-machine-learning

Jupyter Notebooks for *Ensemble Methods for Machine Learning*
https://github.com/gkunapuli/ensemble-methods-notebooks

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**

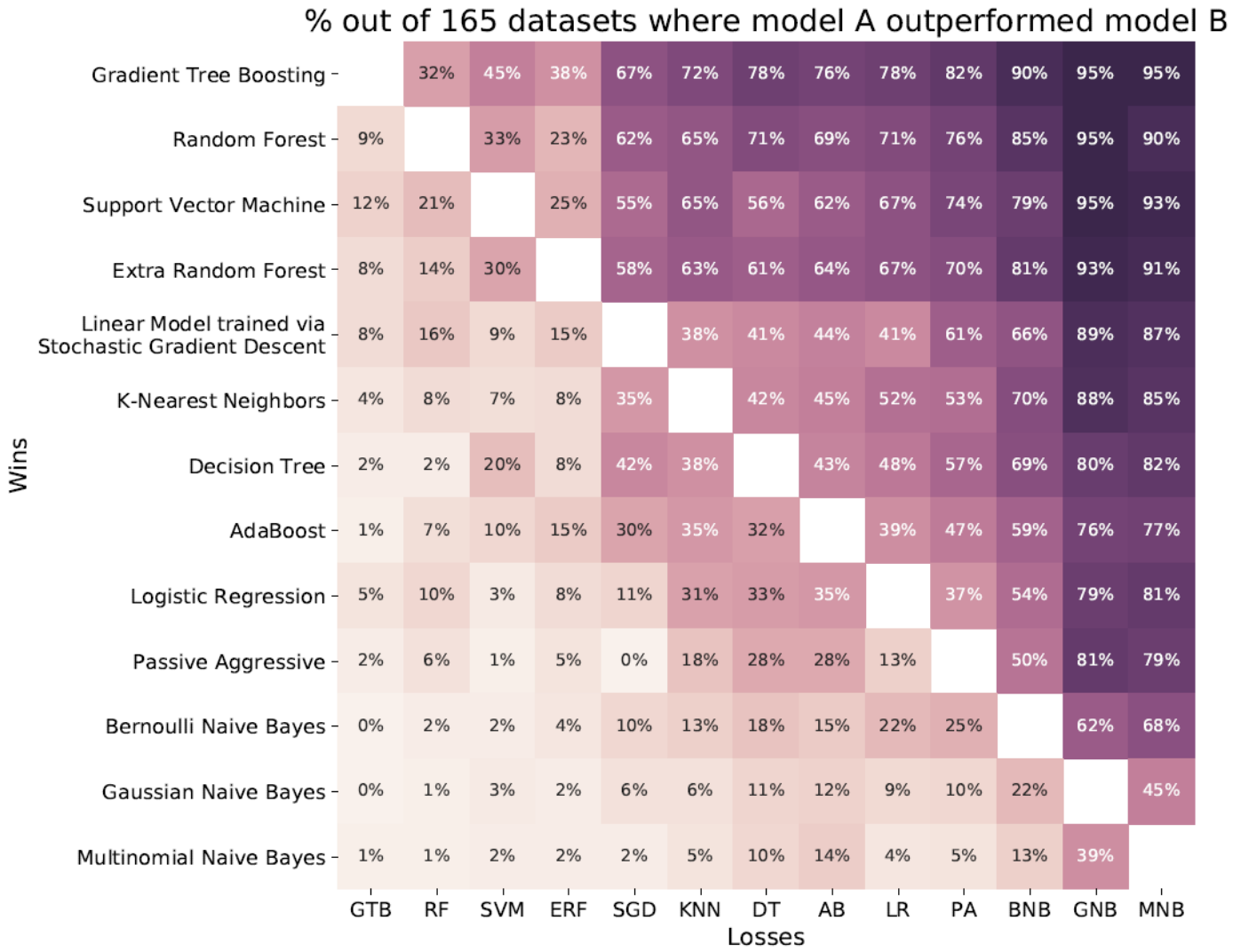
Table 4. Five ML algorithms and parameters that maximize coverage of the 165 benchmark datasets. These algorithm and parameter names correspond to their scikit-learn implementations.

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

[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

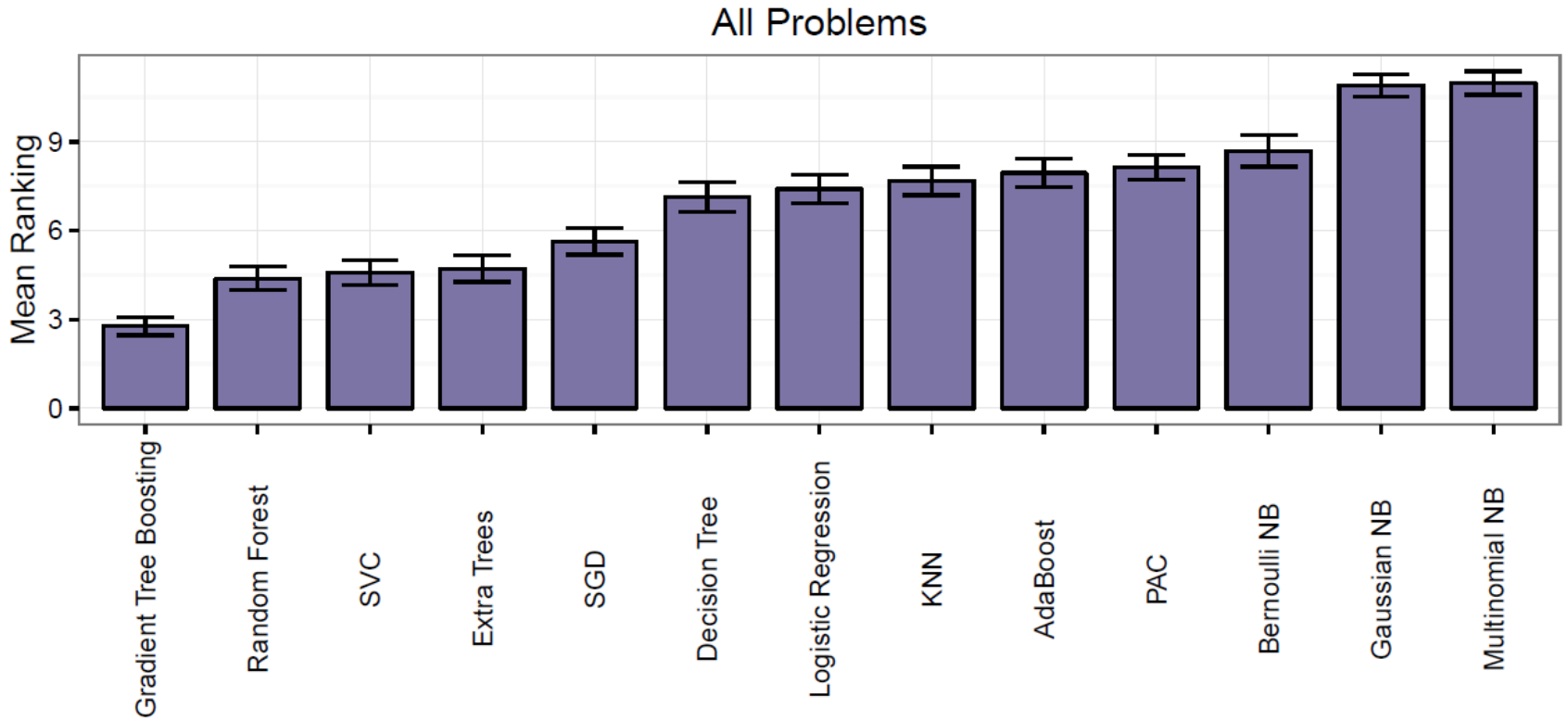
Gradient Boosting: State of the Art



[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.

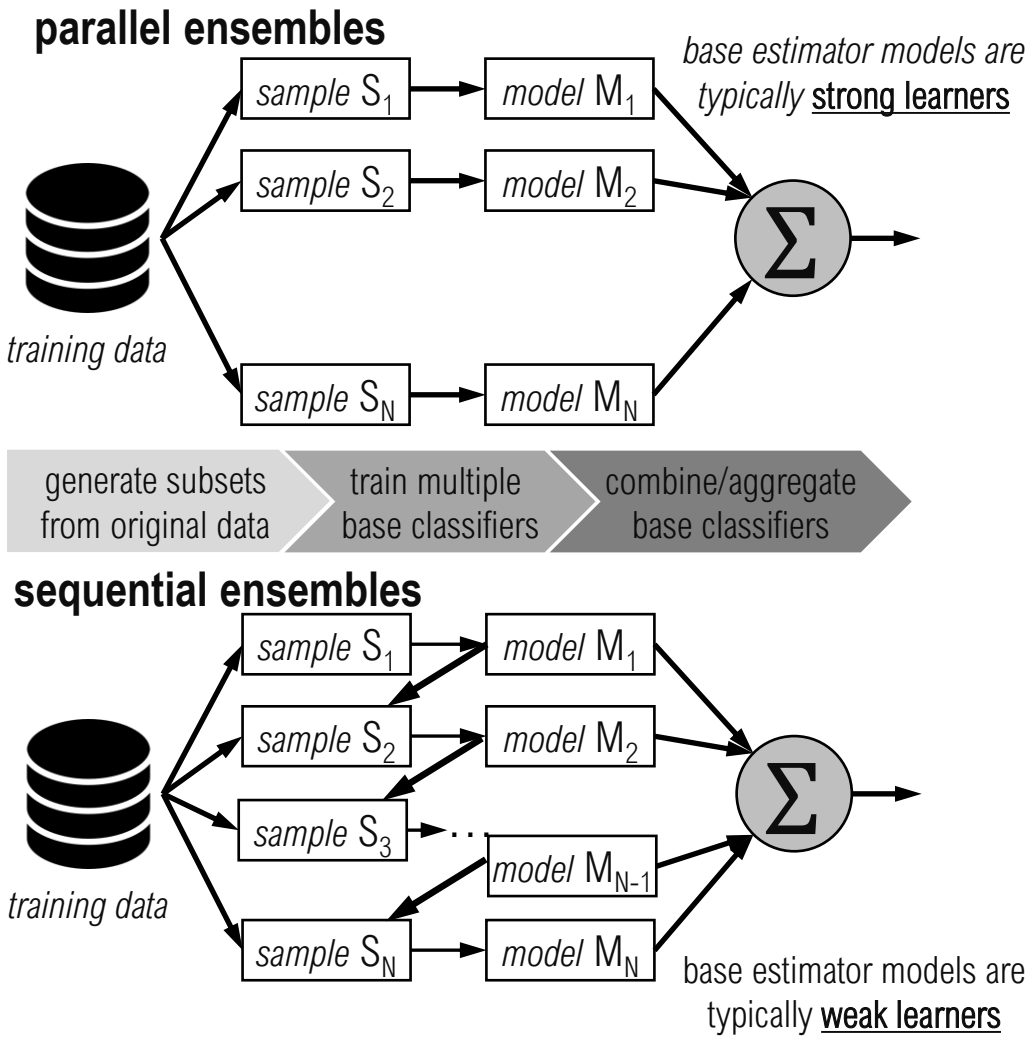
Gradient Boosting: State of the Art



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

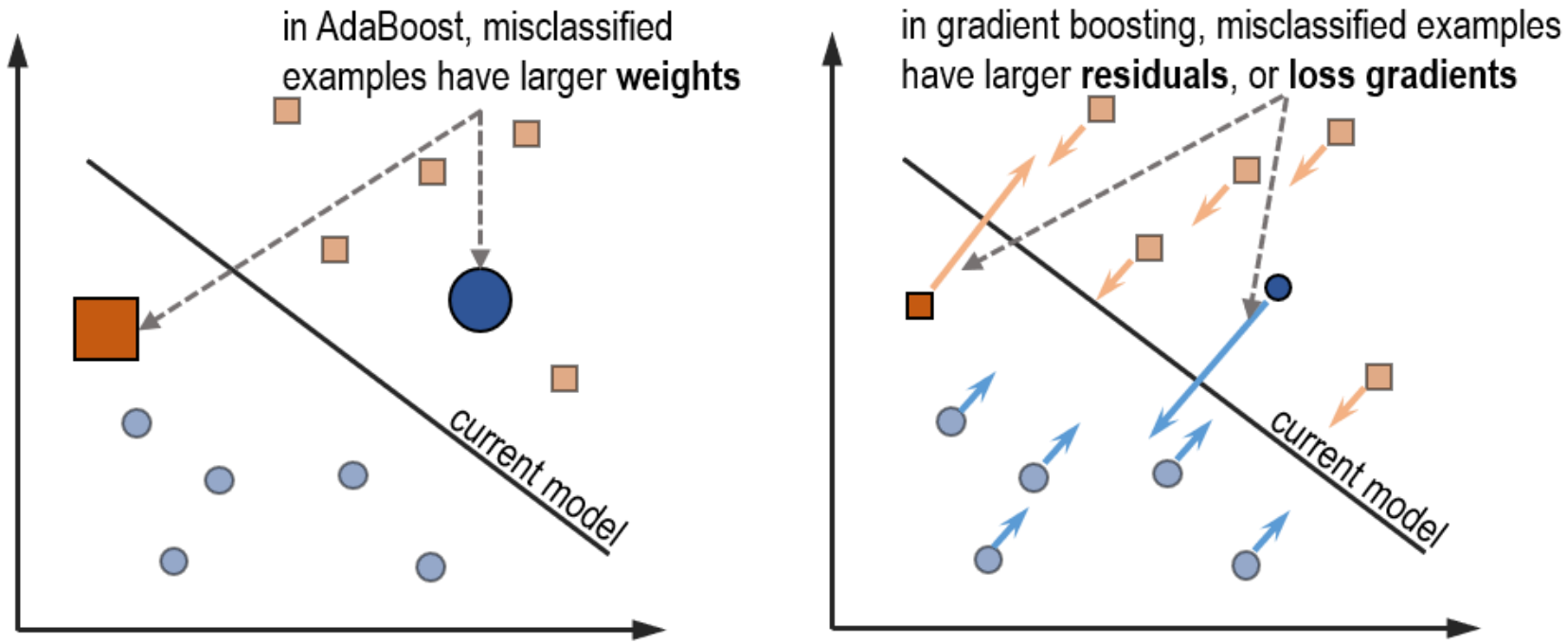
Highly Bothersome, Yet Utterly Unavoidable Question: Which machine learning algorithm should I use for this data set?

Gradient Boosting is a Sequential Ensemble Method



- **sequential ensemble methods** train a weak estimator at each iteration to **fix the errors** made by the weak estimator at **the previous iteration**
- Examples of **parallel ensemble methods** include **Bagging** and **Random Forest**
- Examples of **sequential ensemble methods** include boosting algorithms such as **AdaBoost** and **Gradient Boosting**
- Sequential ensemble methods need to identify training examples that are badly misclassified to tell the base learning algorithm it should focus on those examples in current iteration

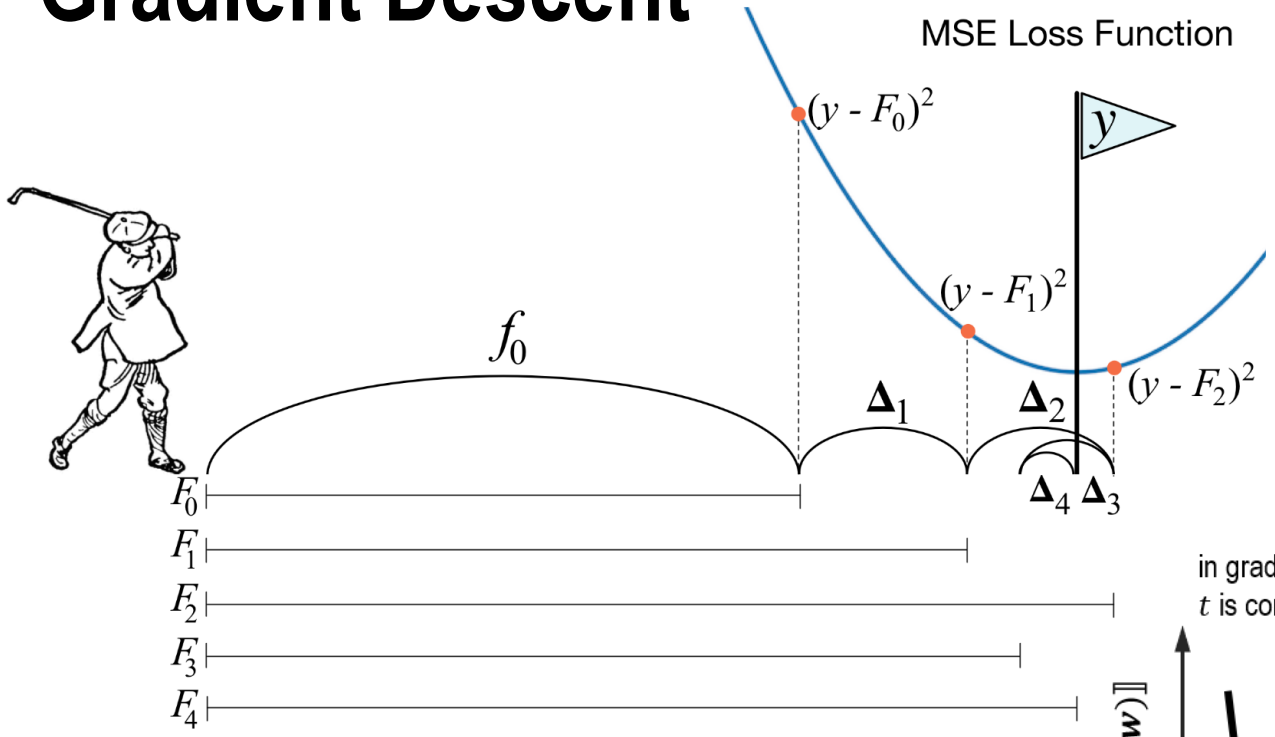
Gradient Boosting vs. AdaBoost



AdaBoost and gradient boosting identify **high-priority misclassified examples** in different ways:

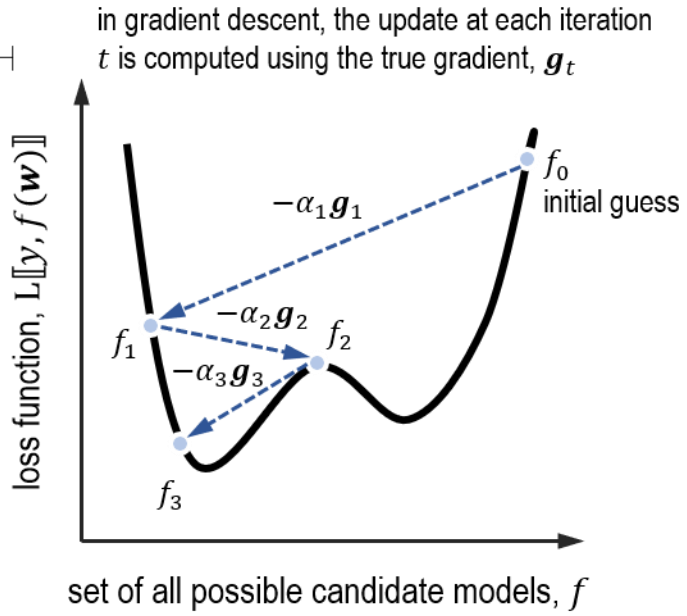
- AdaBoost identifies such examples by **weighting**
  - misclassified examples have higher weights than correctly classified ones;
- Gradient boosting uses **residuals** or **errors** (between the true and predicted labels)
  - residuals can be computed directly from the loss function

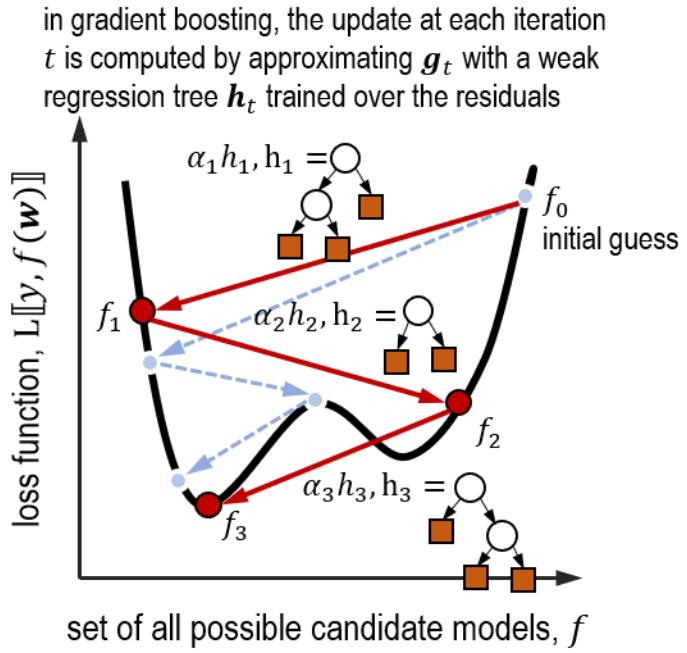
Gradient Boosting Approximates Gradient Descent



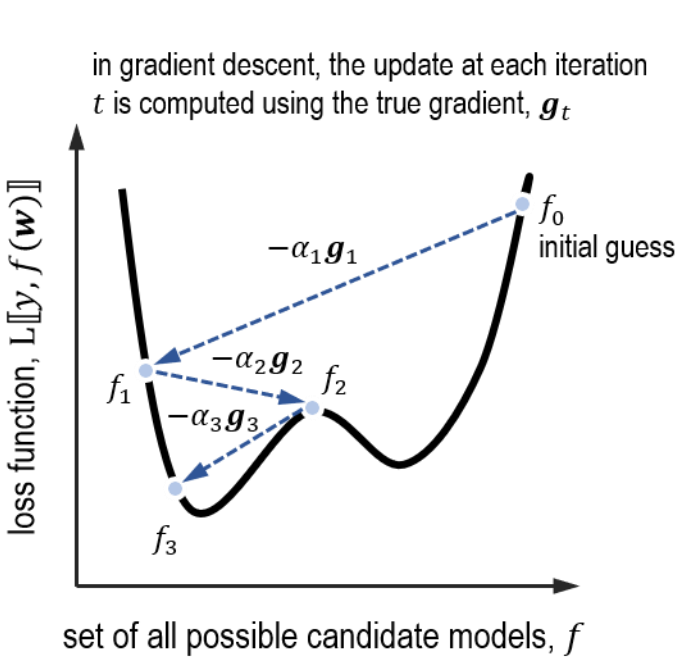
Main intuition: descent algorithms take <u>sequential</u> steps to get to a minimizer;

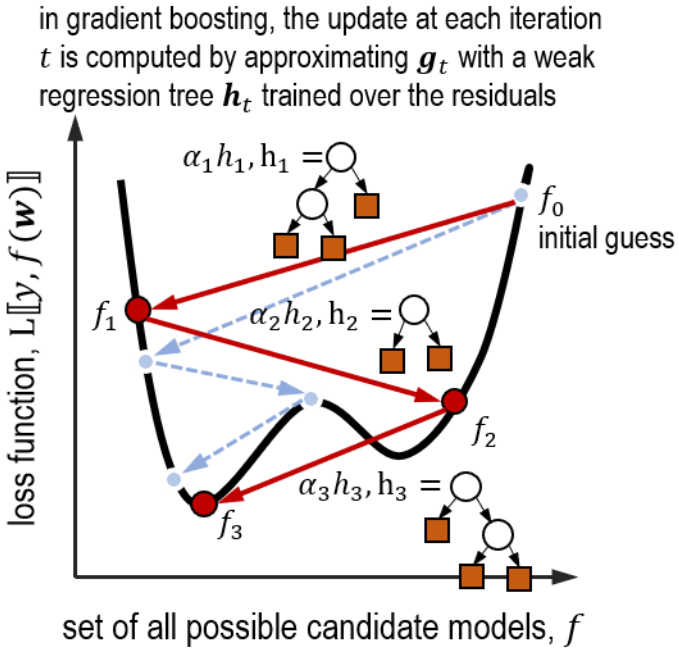
- in gradient descent, we **explicitly compute the gradient**; for loss functions this is the total residual ($\boldsymbol{g_t}$)
- in gradient boosting, we **learn an approximate gradient** by a weak learner to fit the residuals ($\boldsymbol{h_t}$)





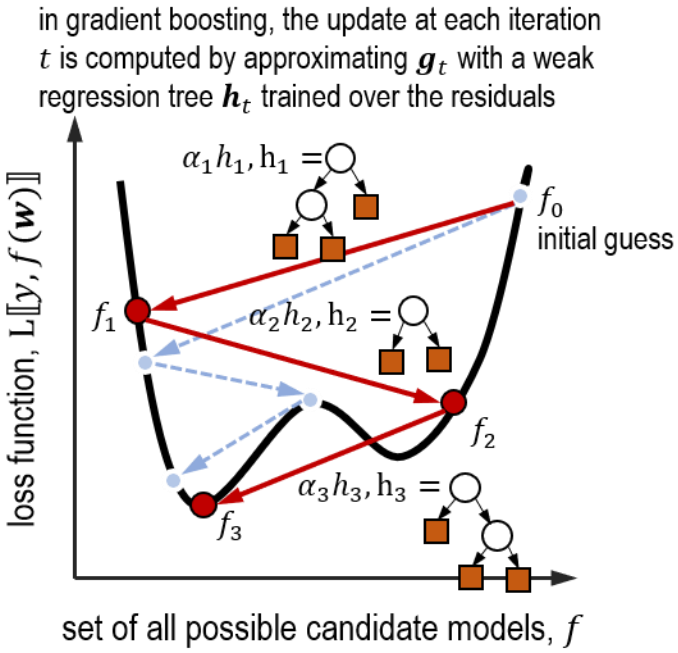
Gradient Boosting = Gradient Descent + Boosting





- *Like AdaBoost*, gradient boosting trains a weak learner to fix the mistakes made by the previous weak learner.
  - Adaboost uses weights over training examples to train weak learners in the ensemble
  - Gradient boosting uses **residuals of the training examples**
- *Like gradient descent*, gradient boosting updates the current model with gradient information
  - Gradient descent uses the gradient directly
  - Gradient boosting **trains a weak learner** over the residuals to approximate the gradient

Gradient Boosting from Scratch



initialize: $F = f_0$, a constant value

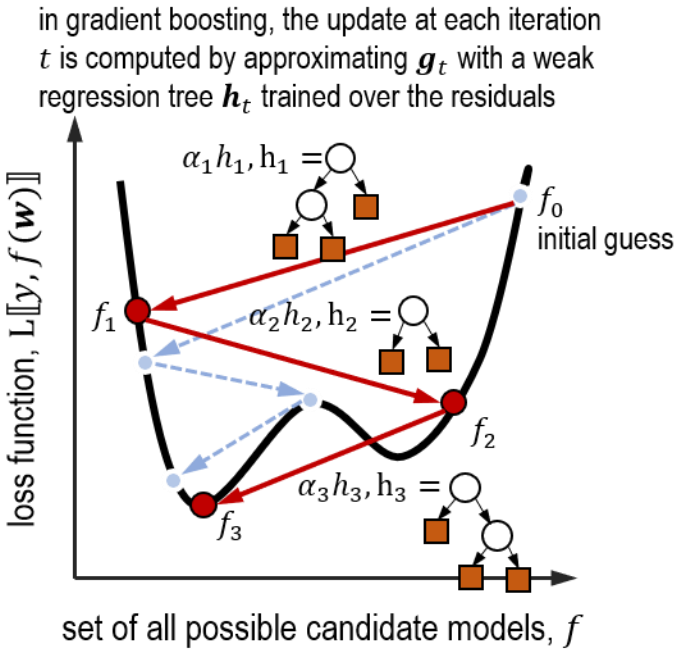
for t = 1 to T:

1. ***compute residuals*** for each training example

$$r_i^t = -\frac{\partial L}{\partial F}(y_i, y)$$

2. ***fit a weak learner*** $\boldsymbol{h_t(x)}$ over training examples and their corresponding residuals $(\boldsymbol{x}_i, r_i)_{i=1}^n$
3. ***compute step length*** (α_t) using line search
4. ***update the model*** $F = F + \alpha_t \cdot \boldsymbol{h_t(x)}$

Gradient Boosting from Scratch



initialize: $F = f_0$, a constant value

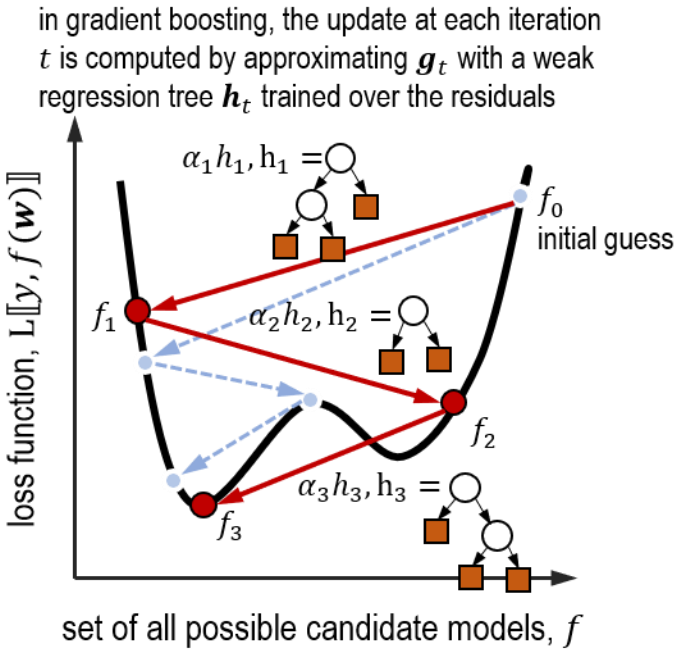
for t = 1 to T:

1. ***compute residuals*** for each training example

$$r_i^t = -\frac{\partial L}{\partial F}(y_i, y_i^{\text{pred}})$$

2. ***fit a weak learner*** $\boldsymbol{h_t(x)}$ over training examples and their corresponding residuals $(\boldsymbol{x}_i, r_i)_{i=1}^n$
3. ***compute step length*** (α_t) using line search
4. ***update the model*** $F = F + \alpha_t \cdot \boldsymbol{h_t(x)}$

Gradient Boosting from Scratch



initialize: $F = f_0$, a constant value

for t = 1 to T:

1. ***compute residuals*** for each training example

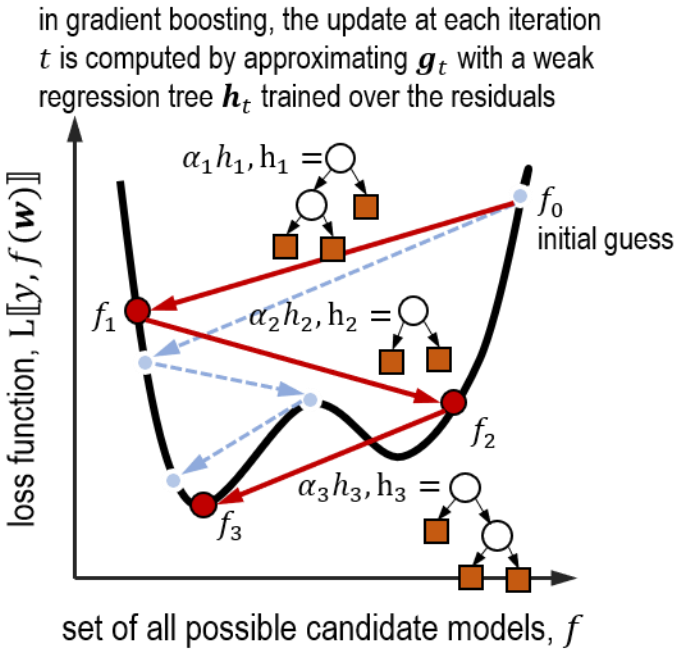
$$r_i^t = -\frac{\partial L}{\partial F}(y_i, y_i^{\text{pred}})$$

2. ***fit a weak learner*** $\boldsymbol{h_t(x)}$ over training examples and their corresponding residuals $(\boldsymbol{x}_i, r_i)_{i=1}^n$
3. ***compute step length*** (α_t) using line search
4. ***update the model*** $F = F + \alpha_t \cdot \boldsymbol{h_t(x)}$

For a single example

- loss function L measures the **error** between true label and predicted label
- (functional) gradient of loss function tells us the **residual** or how much we are off
- **the negative gradient** tells us what we must do to fix the mistake

Gradient Boosting from Scratch



initialize: $F = f_0$, a constant value

for t = 1 to T:

1. ***compute residuals*** for each training example

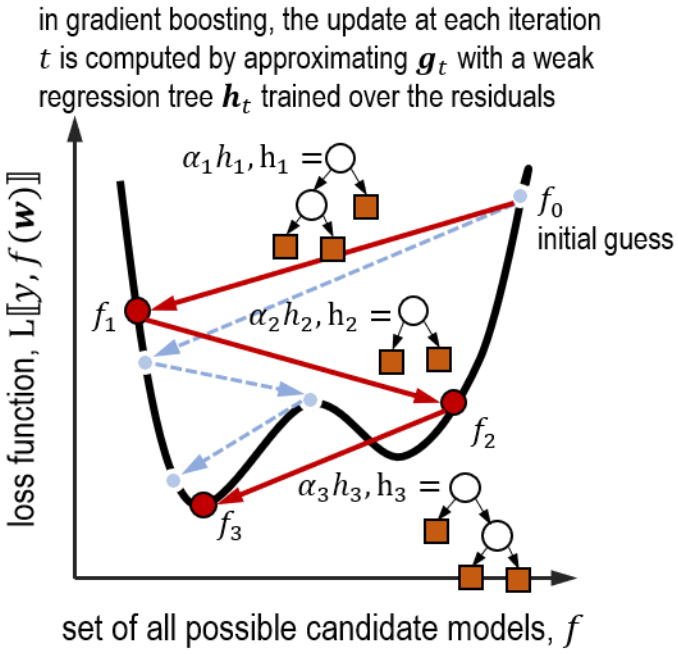
$$r_i^t = -\frac{\partial L}{\partial F}(y_i, y_i^{\text{pred}})$$

2. ***fit a weak learner*** $\boldsymbol{h_t(x)}$ over training examples and their corresponding residuals $(\boldsymbol{x}_i, r_i)_{i=1}^n$
3. ***compute step length*** (α_t) using line search
4. ***update the model*** $F = F + \alpha_t \cdot \boldsymbol{h_t(x)}$

For all examples we have

- the features $(\boldsymbol{x}_i)$ and the residuals (r_i)
- we train a weak learner (typically a decision tree) to the data set $(\boldsymbol{x}_i, r_i)_{i=1}^n$
- since residuals (r_i) are continuous values, this training task is a **regression problem!**
- in fact, gradient boosting's weak learning algorithms are always regression algorithms

Gradient Boosting from Scratch



initialize: $F = f_0$, a constant value

for t = 1 to T:

1. ***compute residuals*** for each training example

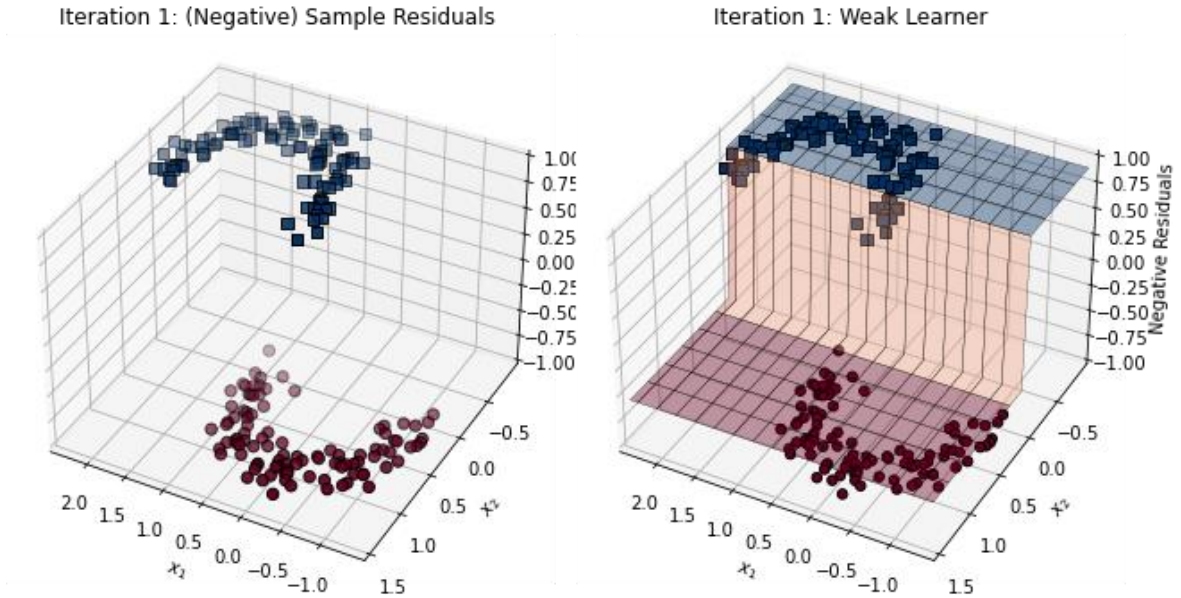
$$r_i^t = -\frac{\partial L}{\partial F}(y_i, y_i^{\text{pred}})$$

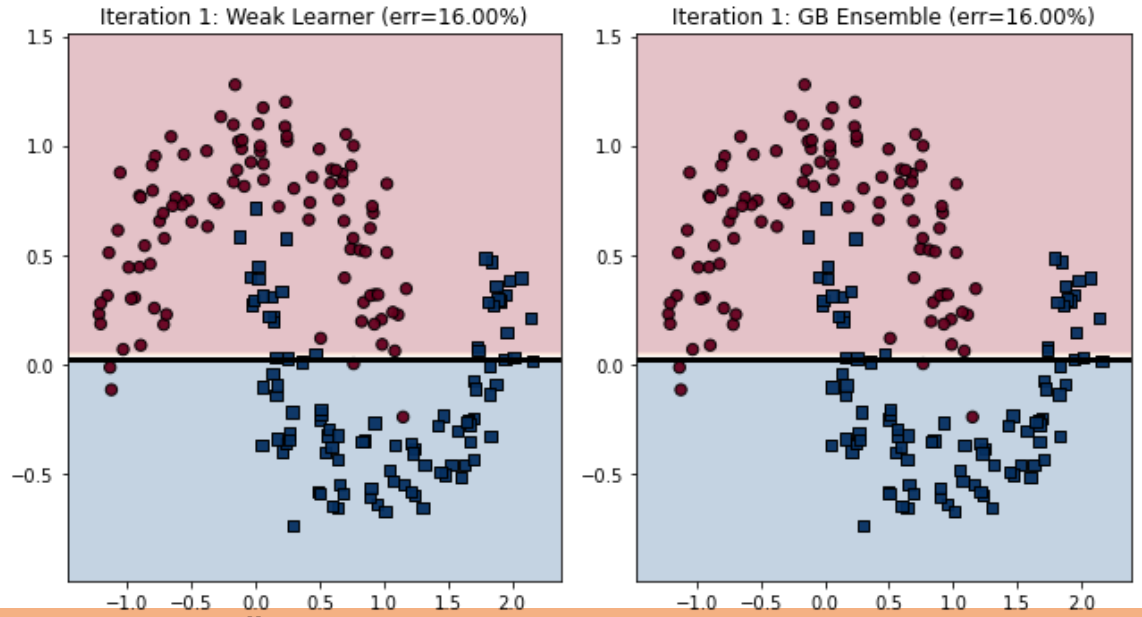
2. ***fit a weak learner*** $\boldsymbol{h}_t(\boldsymbol{x})$ over training examples and their corresponding residuals $(\boldsymbol{x}_i, r_i)_{i=1}^n$
3. ***compute step length*** (α_t) using line search
4. ***update the model*** $F = F + \alpha_t \cdot \boldsymbol{h}_t(\boldsymbol{x})$

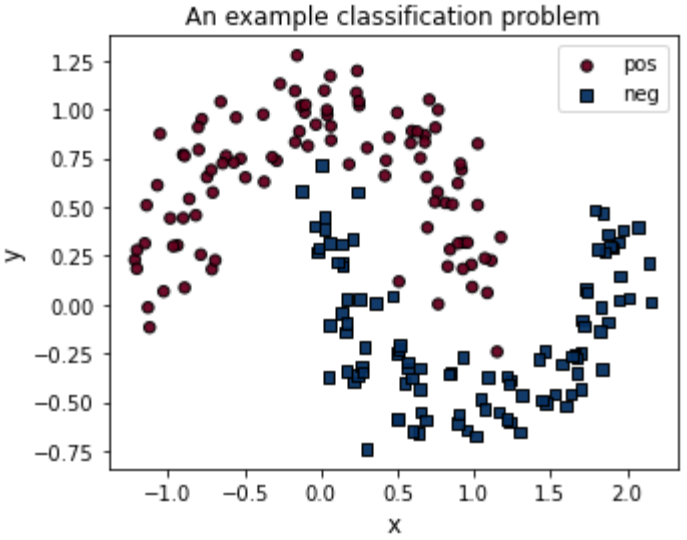
α_t is the weight of weak learner $\boldsymbol{h}_t$ (or the step length)

- it can be computed iteration by iteration by a line search procedure; slow
- it can be user-defined, set $\alpha_t = \eta$ (in the range $0 < \eta \leq 1$); called **learning rate**
  - learning rate must be selected by cross validation

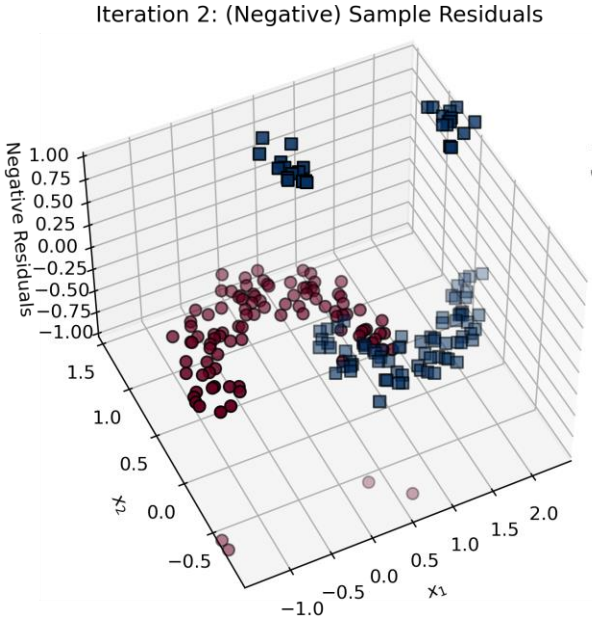
Gradient Boosting Step-By-Step

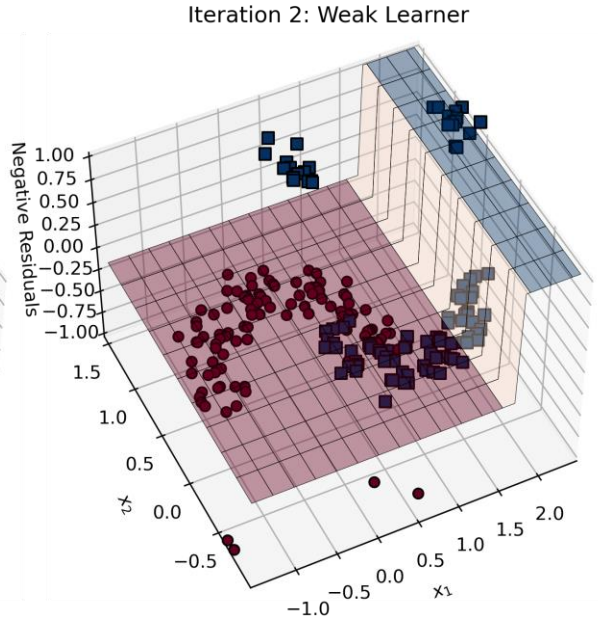


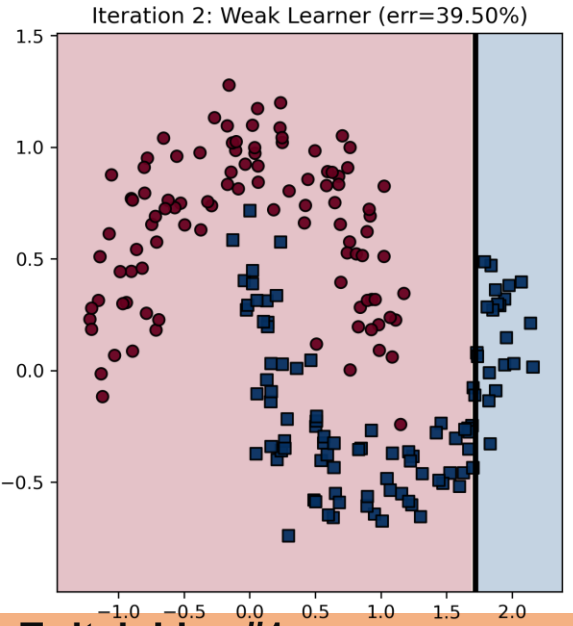


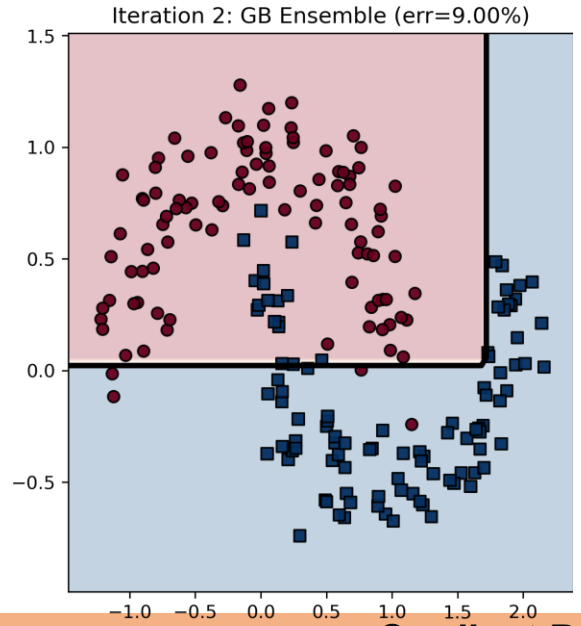


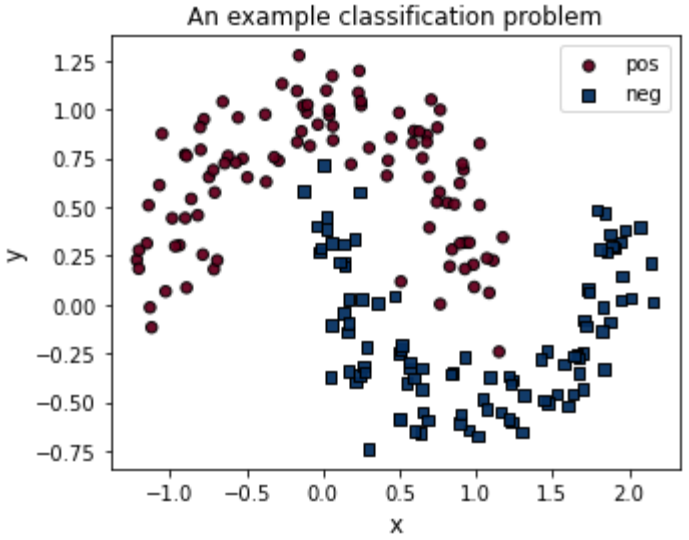
Gradient Boosting Step-By-Step



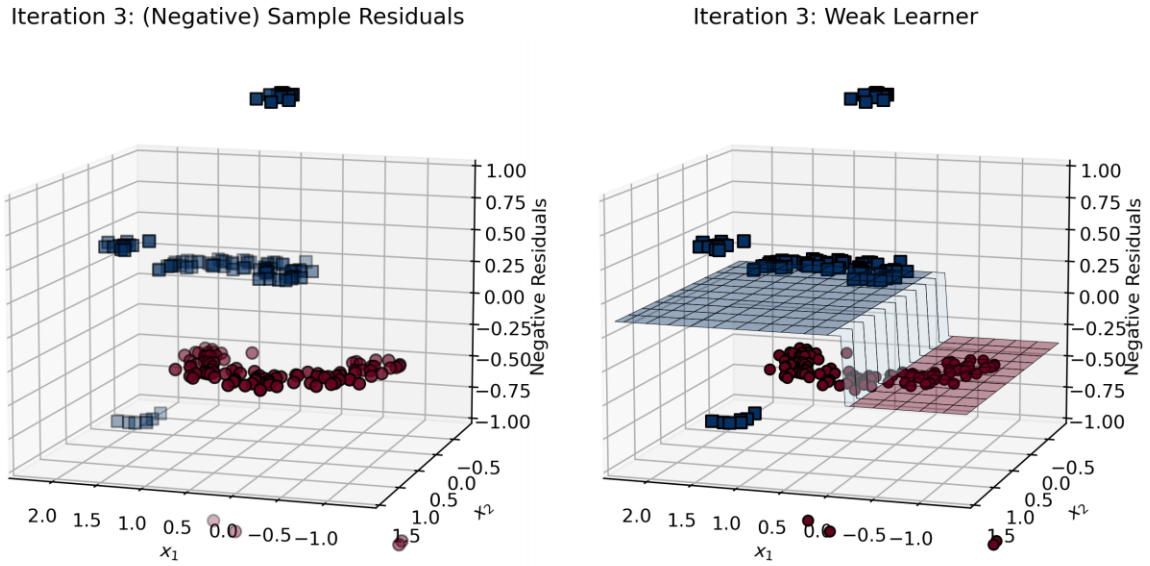


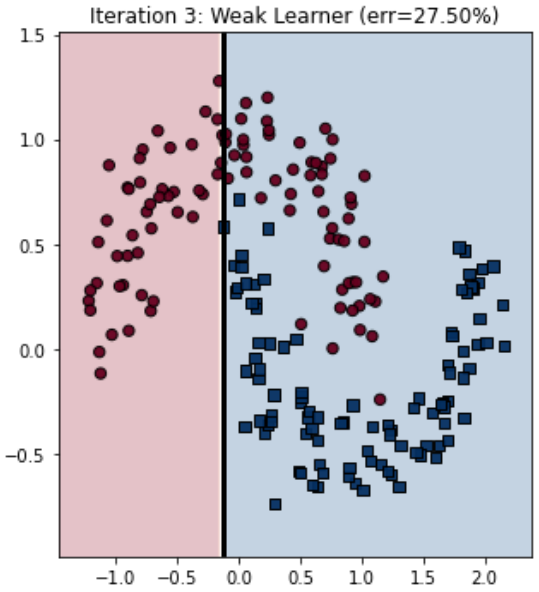


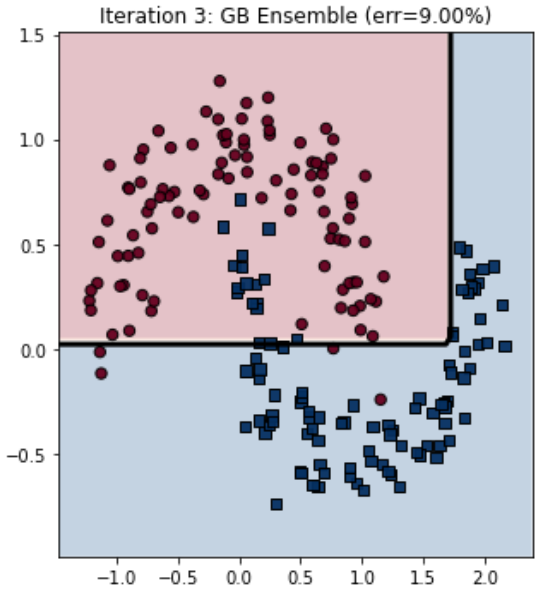


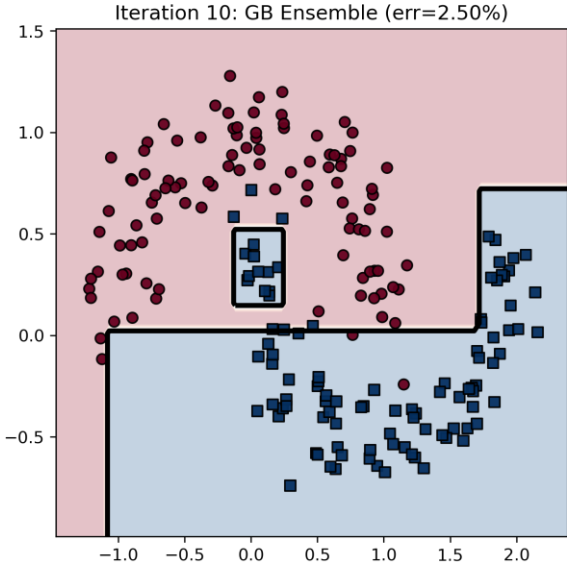


Gradient Boosting Step-By-Step

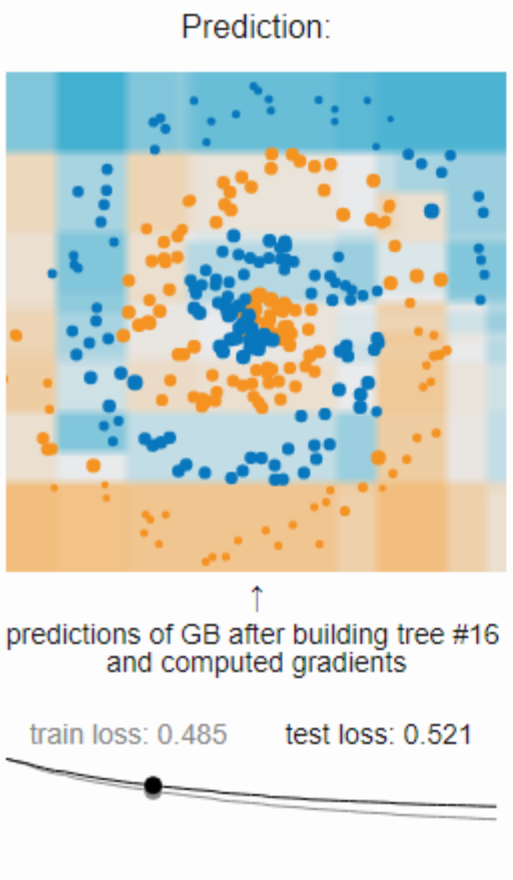


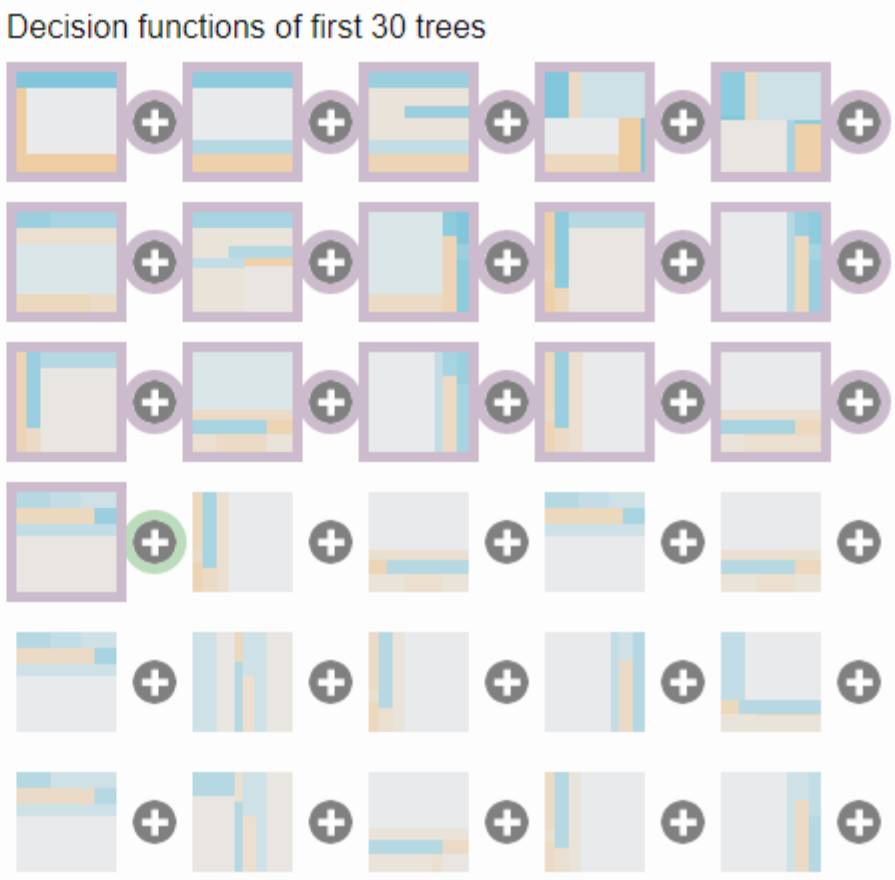






Visualizing Gradient Boosting Further





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

LightGBM: Light Gradient Boosted Machines

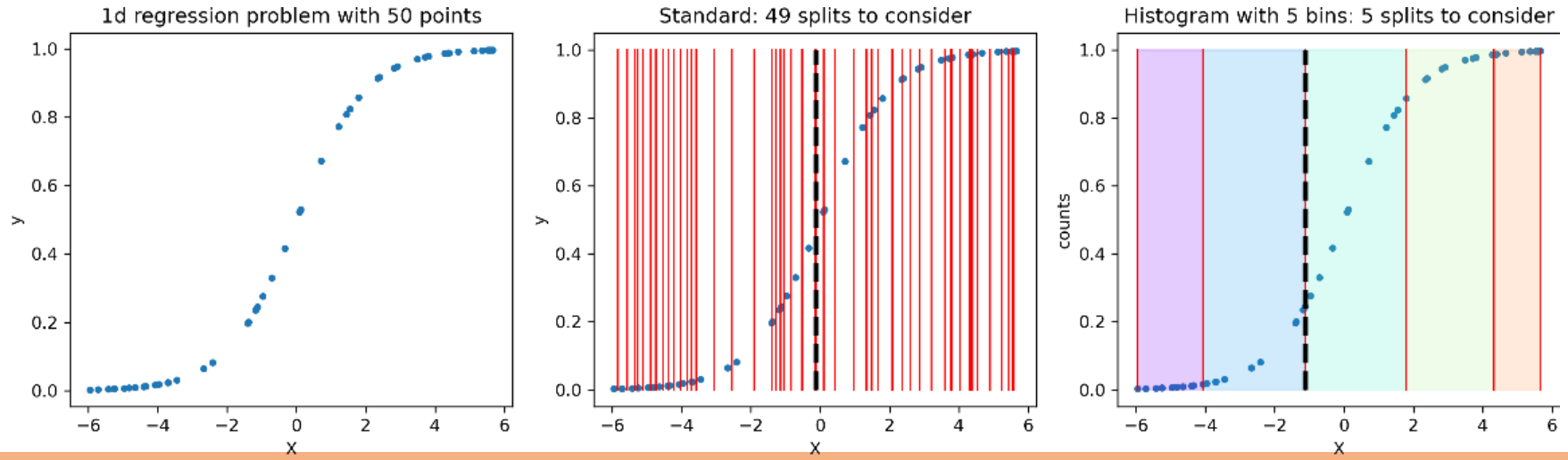
- Open-source gradient boosting framework that was originally developed and released by Microsoft
  - Covered in Chapter 5

- **Histogram-based gradient boosting** with algorithmic speedups such as **gradient based one-sided sampling** and **exclusive feature bundling**
  - faster training and lower memory usage

- Support for a large number of loss functions for classification, regression and ranking
  - can also create and use application-specific custom loss functions

- Support for parallel and GPU learning
  - not covered in this book

Standard tree learning evaluates every possible split (center) to find the best split. Histogram-based binning first puts the data into buckets, and then evaluates the splits between each pair of data buckets.

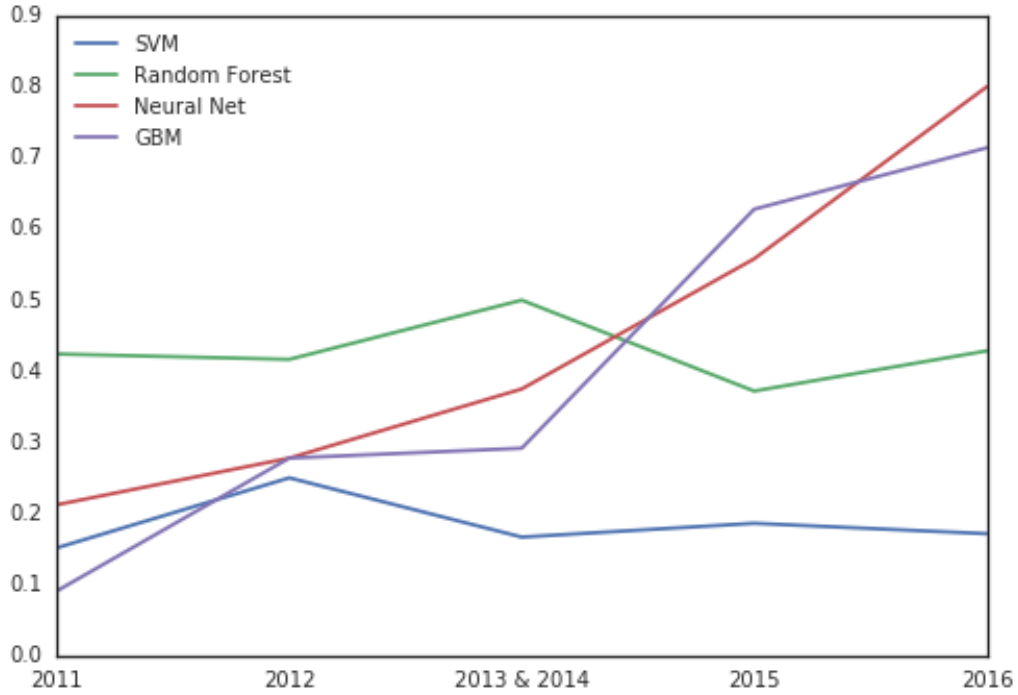


XGBoost: Extreme Gradient Boosting

- **Open-source gradient boosting framework**
  - "wins" every Kaggle competition
  - Covered in Chapter 6
- **Newton Boosting**
  - uses **second-order derivative** information to speed up convergence
- **Regularization**
  - tree complexity is penalized
  - gradients computed for regularized loss functions
- Support for large number of loss functions for classification, regression and ranking
  - can also create and use application-specific custom loss functions
- Support for parallel and GPU learning
  - not covered in this book

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.



Manning page for *Ensemble Methods for Machine Learning* (in MEAP)
https://www.manning.com/books/ensemble-methods-for-machine-learning

Jupyter Notebooks for *Ensemble Methods for Machine Learning*
https://github.com/gkunapuli/ensemble-methods-notebooks