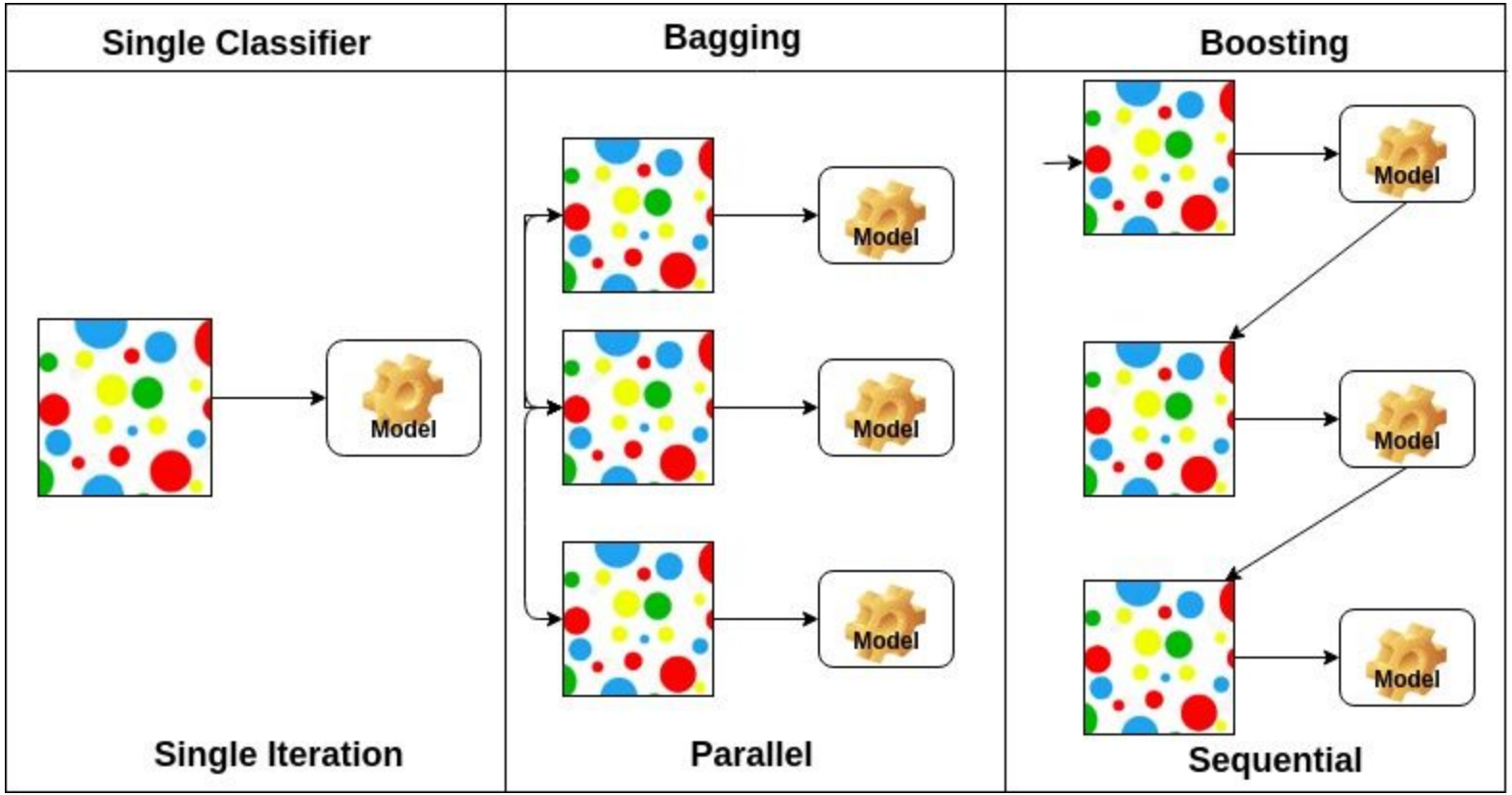
Boosting Algorithms

that are tree-based

Tomasz Kajdanowicz



ex. Random Forest

Sequential Decision Tree Building

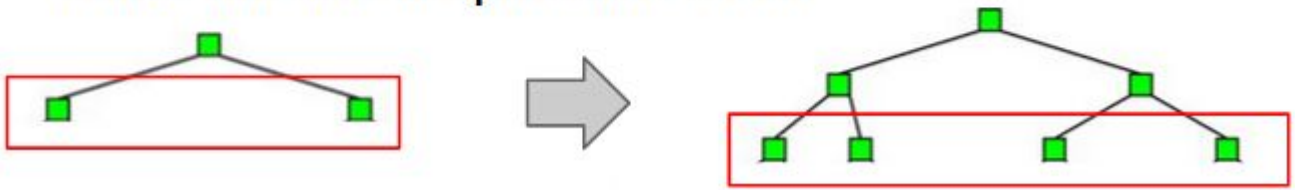
The building process

For each leaf node:

For each feature:

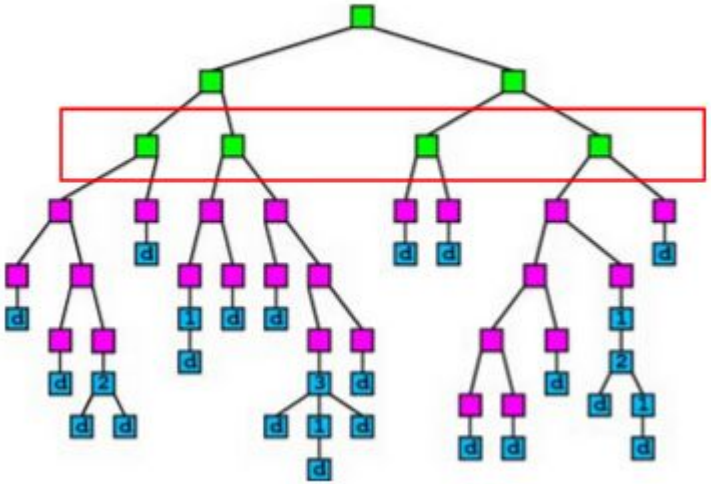
- Sort the instances in the node by the feature value
- Linear scan to decide the best split on the feature

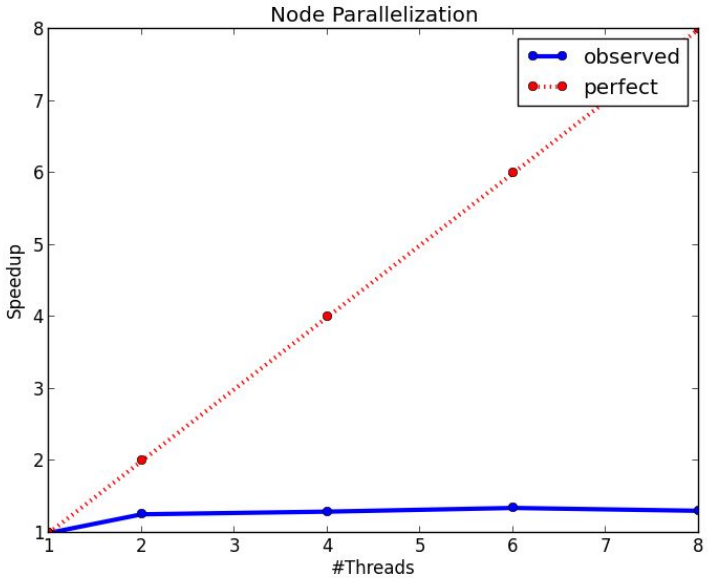
Take the best split and do it



Parallelize Node Building at Each Level

- parallelize node building at each level
- **workload imbalance** problem (number of leafs)

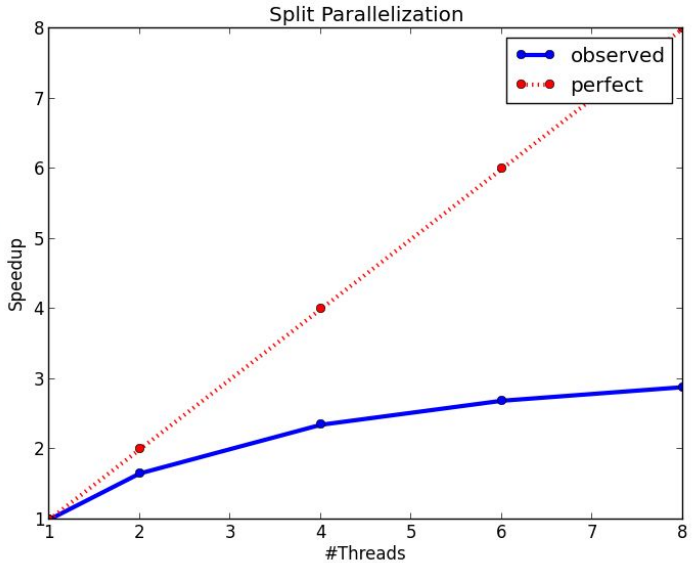




Parallelize Split Finding on Each Node

- too much overhead for small nodes

1. For each feature:
   - Sort the instances by the feature value
   - Linear scan to decide the best split on the feature
2. Take the best split and do it



Parallelize Split Finding at Each Level by Features

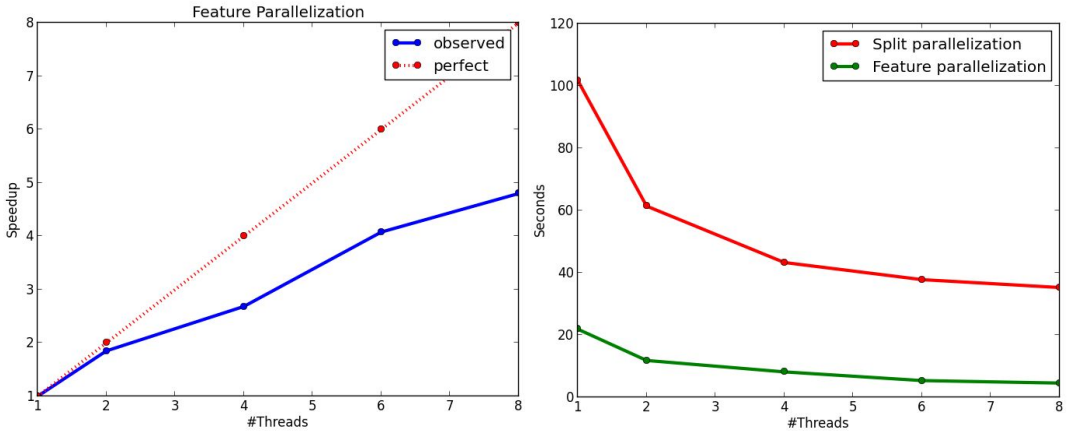
For each feature:

For each leaf node:

Can sort globally first!

- ~~Sort the instances of the node by the feature value~~
- Linear scan to decide the best split on the feature

Take the best split for each node and do it



Advantages:

- workload totally balanced - number of instances for each feature is the same, the workload for different jobs is the same
- overhead for parallelization is small - split finding at the whole level rather than a single node

AdaBoost
AdaBoost.PL

Algorithm 1 ADABOOST(D_n, T)

Input: Training set of n examples (D_n)
Number of boosting iterations (T)
Output: The classifier (H)
Procedure:

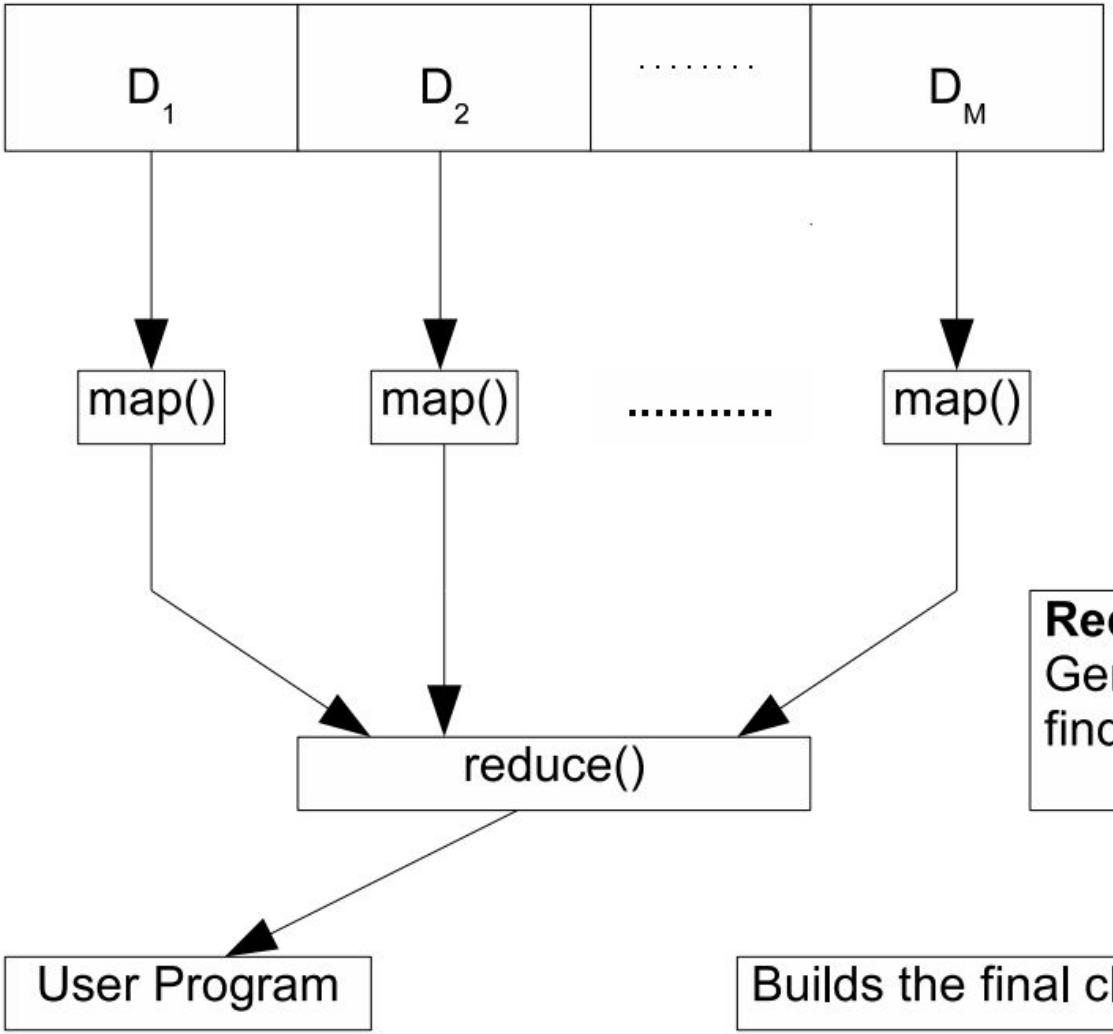
$w^1 \leftarrow \left(\frac{1}{n}, ..., \frac{1}{n}\right)$
for $t \leftarrow 1$ to T **do**
  $h^{(t)} \leftarrow$ LEARNWEAKCLASSIFIER(w^t)
  $\epsilon_- \leftarrow \sum_{i=1}^{n} w_i^t I\left\{h^{(t)}(x_i) \neq y_i\right\}$
  $\alpha^t \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon_-}{\epsilon_-}\right)$
  for $i \leftarrow 1$ to n **do**
    if $h^{(t)}(x_i) \neq y_i$ **then**
      $w_i^{t+1} \leftarrow \frac{w_i^t}{2\epsilon_-}$
    else
      $w_i^{t+1} \leftarrow \frac{w_i^t}{2(1-\epsilon_-)}$
    end if
  end for
end for
return $H = \sum_{t=1}^{T} \alpha^t h^{(t)}$

Algorithm 2 ADABOOST.PL($D_{n^1}^1$, ... , $D_{n^M}^M$, T)

Input: The training sets of M workers ($D_{n^1}^1, ..., D_{n^M}^M$)
Number of boosting iterations (T)
Output: The classifier (H)
Procedure:

for $p \leftarrow 1$ to M **do**
  $H^p \leftarrow$ ADABOOST$(D_{n^p}^p, T)$
  $H^{p^*} \leftarrow$ the weak classifiers in H^p sorted w.r.t. $\alpha^{p(t)}$
end for
for $t \leftarrow 1$ to T **do**
  $h^{(t)} \leftarrow$ MERGE$\left(h^{1^*(t)}, ..., h^{M^*(t)}\right)$
  $\alpha^t \leftarrow \frac{1}{M} \sum_{p=1}^{M} \alpha^{p^*(t)}$
end for
return $H = \sum_{t=1}^{T} \alpha^t h^{(t)}$

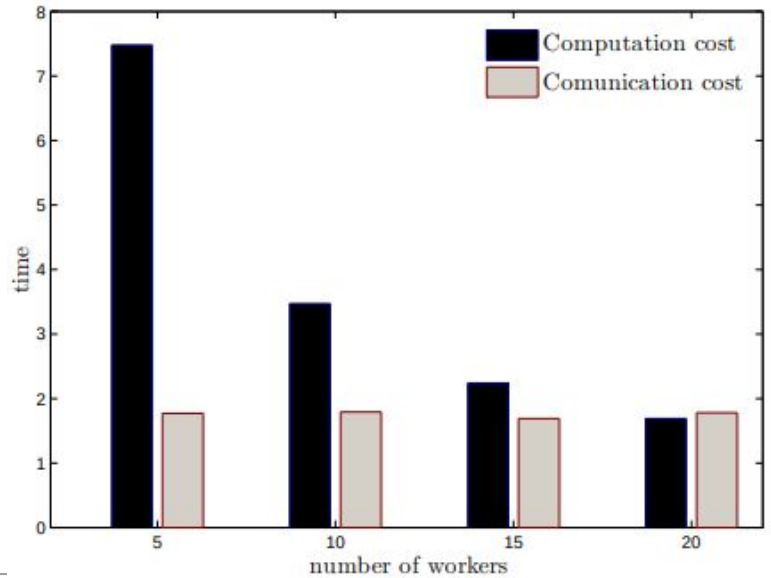
Palit, Indranil, and Chandan K. Reddy. "Parallelized boosting with map-reduce." *2010 IEEE International Conference on Data Mining Workshops*. IEEE, 2010.

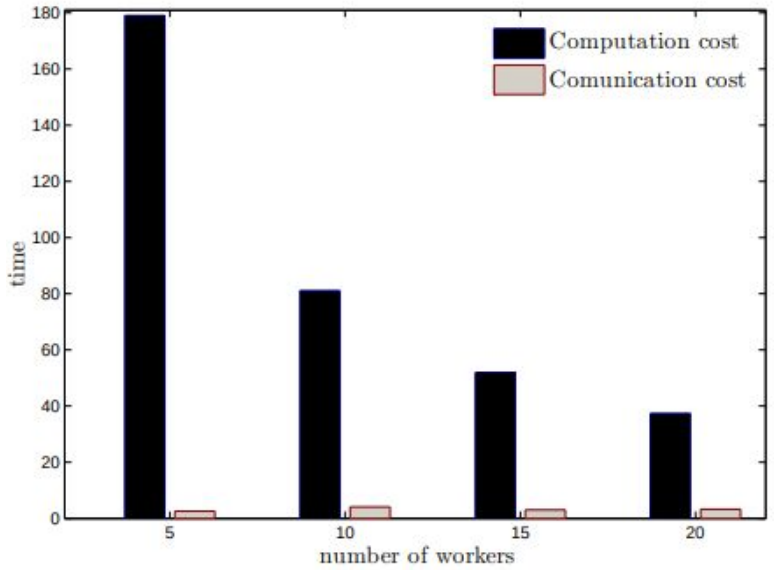


Map task -
Runs AdaBoost on its data and returns the sorted list of weak classifiers.

Reduce Task -
Generates the merged classifier and finds its weight.

Builds the final classifier from Reducer output.





dataset	instances	attributes
musk	6598	167
swsequence	3527	6349

Gradient boosting machines (GBMs)

How to understand GBMs?

additive modeling

intuition behind gradient boosting

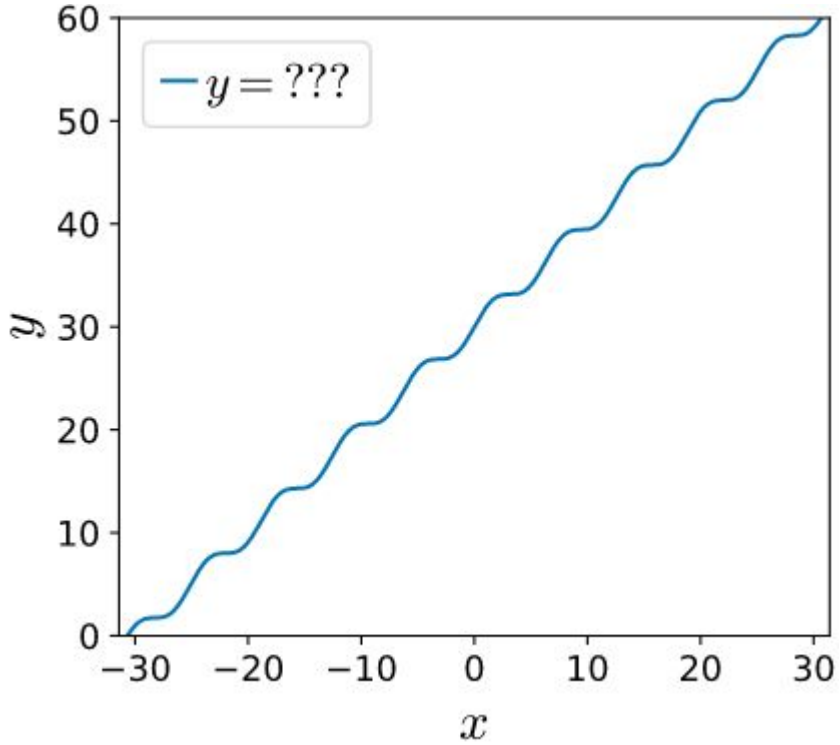
Example

Performance

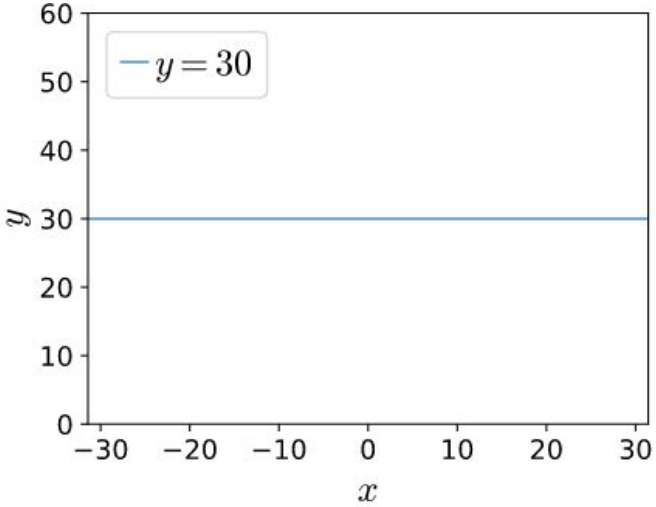
Hyper-parameters

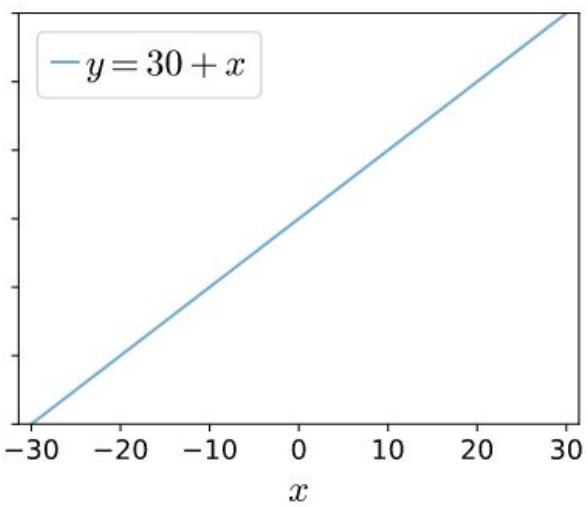
L2 loss optimizing algorithm

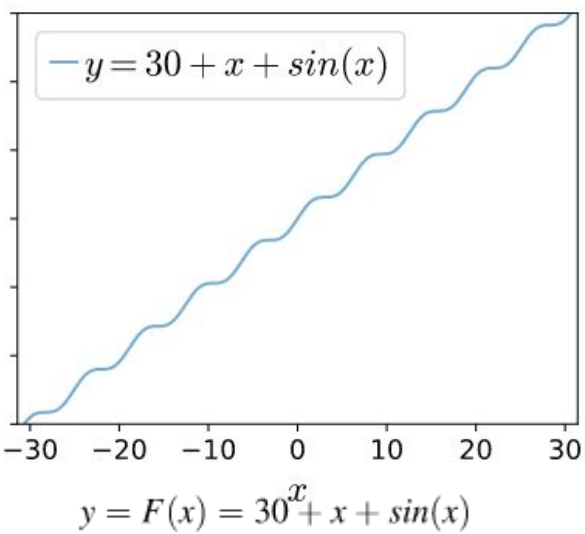
Additive modeling



Additive modeling







$$F_M(\mathbf{x}) = f_1(\mathbf{x}) + \ldots + f_M(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x})$$

Greedy approach

Our aim: $\hat{y} = \sum_{m=1}^{M} f_m(\mathbf{x})$

Gradient boosting machines use additive modeling to gradually nudge an approximate model with greedy approach:

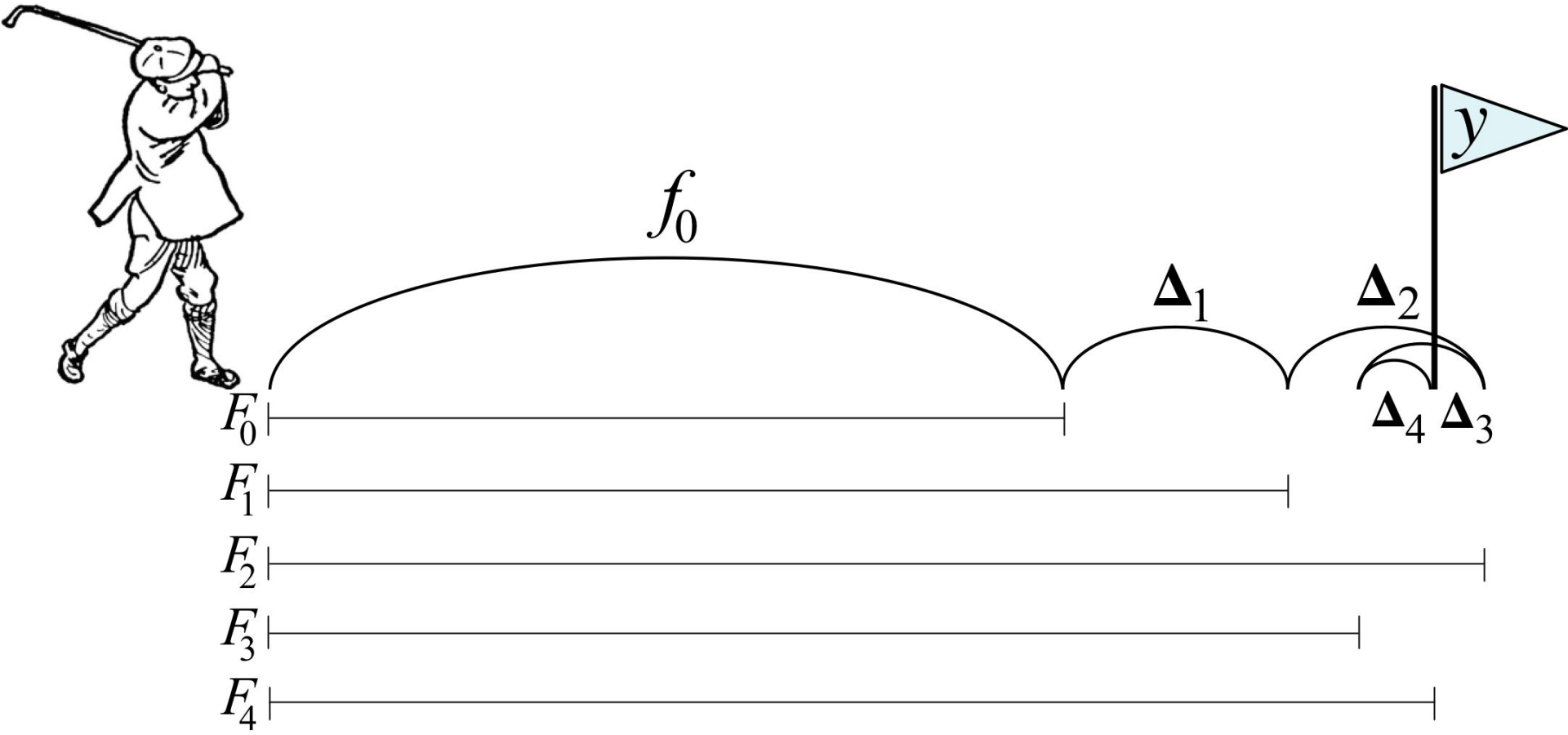
$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + f_m(\mathbf{x})$$

Intuition behind gradient boosting

$f_0(x)$ - initial model (predicts *y* given *x*)

then, let's gradually nudge the model towards the known target value y by adding one or more tweaks

$$\begin{aligned}\hat{y} &= f_0(\mathbf{x}) + \Delta_1(\mathbf{x}) + \Delta_2(\mathbf{x}) + \ldots + \Delta_M(\mathbf{x}) \\ &= f_0(\mathbf{x}) + \sum_{m=1}^{M} \Delta_m(\mathbf{x}) \\ &= F_M(\mathbf{x})\end{aligned}$$


f_0
Δ_1
Δ_2
y
Δ_4 Δ_3
F_0
F_1
F_2
F_3
F_4

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	$\Delta_4 = 5$

REMARK!

- **Direction** vector and **residuals** vector
- There will be so-called **learning rate**, that speeds up or slows down the overall approach of predicted y to real y, which helps to reduce the likelihood of overfitting

Example

Data:

sqfeet	rent
750	1160
800	1200
850	1280
900	1450
950	2000

Notation: $\boldsymbol{X}$ - feature vectors, $\boldsymbol{y}$ - rent vector, $F_m(\boldsymbol{x}_i)$ - predicted value, $F_m(\boldsymbol{X})$ - predicted target vector

Task: predict rent price given square footage

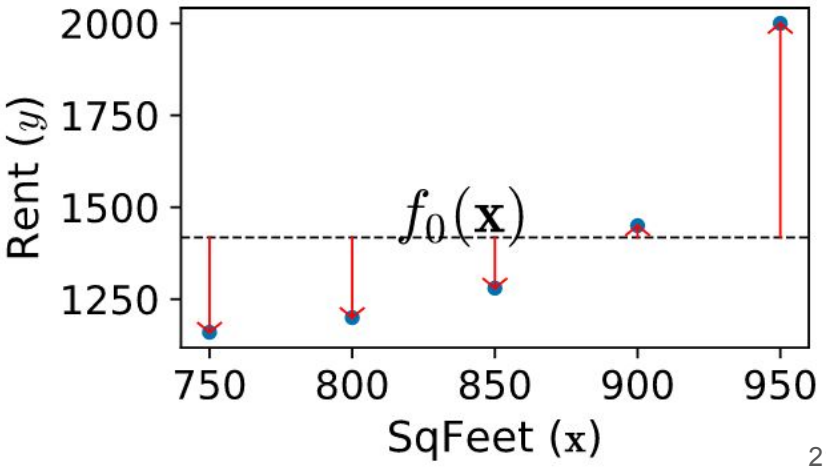
Initial model: mean (average), then: regression tree stump

Example

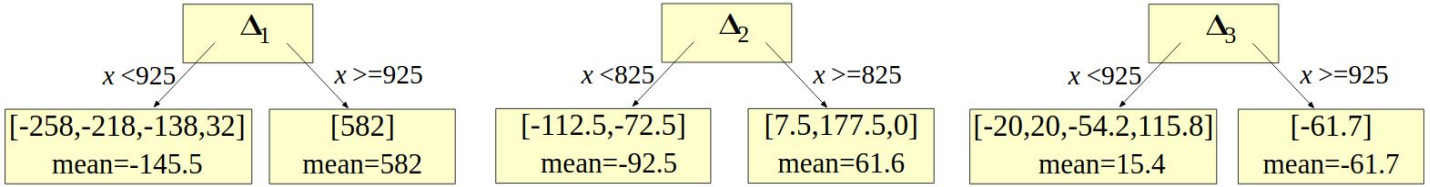
sqfeet	rent	F_0	$\mathbf{y} - F_0$
750	1160	1418	-258
800	1200	1418	-218
850	1280	1418	-138
900	1450	1418	32
950	2000	1418	582

y - F_m - called pseudo-responses

shows not only the direction but the magnitude



Example



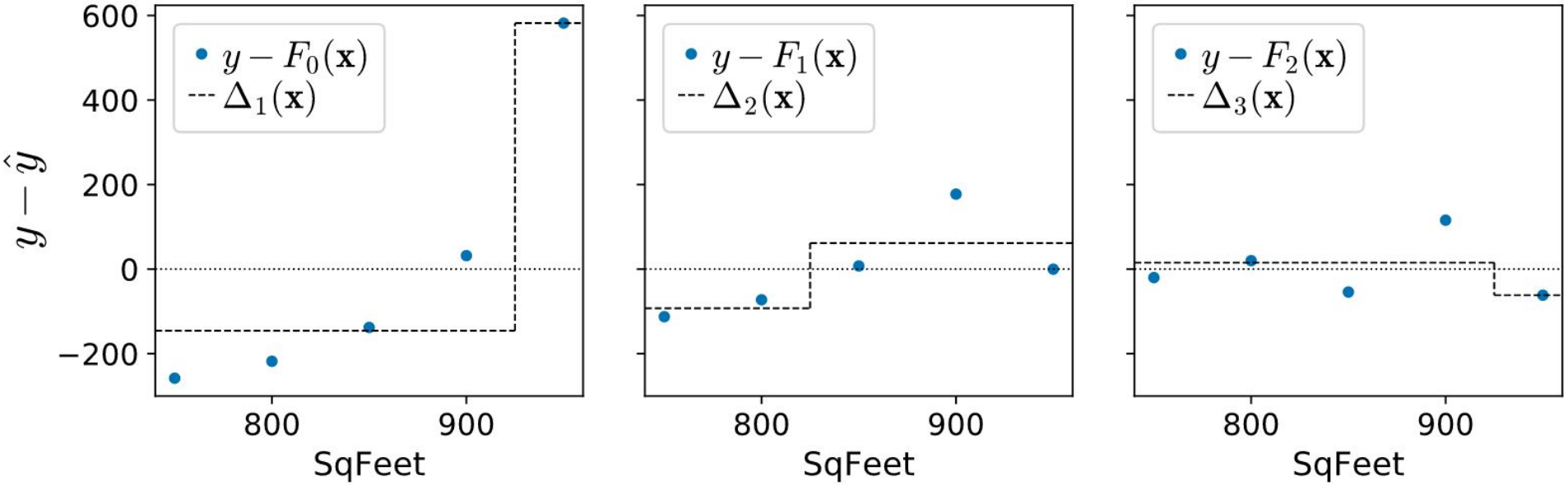
$$F_m(X) = F_{m-1}(X) + \eta \Delta_m(X) \qquad \eta = 1.0$$

$$F_1 = F_0 + \Delta_1, F_2 = F_1 + \Delta_2$$

Δ_1	F_1	y-F_1	Δ_2	F_2	y - F_2	Δ_3	F_3
-145.5	1272.5	-112.5	-92.5	1180	-20	15.4	1195.4
-145.5	1272.5	-72.5	-92.5	1180	20	15.4	1195.4
-145.5	1272.5	7.5	61.7	1334.2	-54.2	15.4	1349.6
-145.5	1272.5	177.5	61.7	1334.2	115.8	15.4	1349.6
582	2000	0	61.7	2061.7	-61.7	-61.7	2000

Example

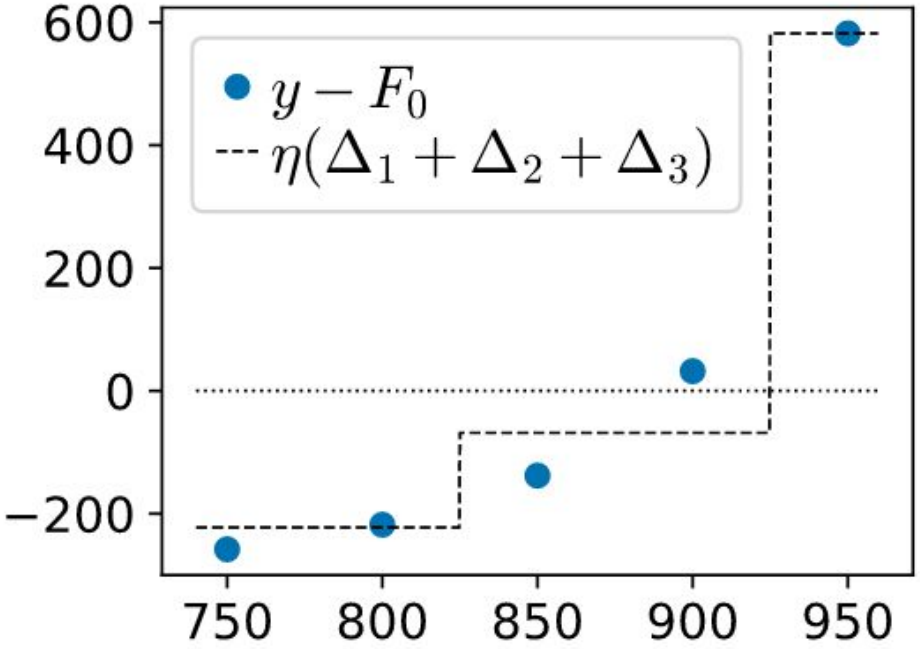
we are always training on the residual vector ***y - F_{m-1}*** but get imperfect model



blue dots - residual vector elements used to train weak models, **dashed lines** - predictions made by weak models, **dotted line** - origin at 0

Example

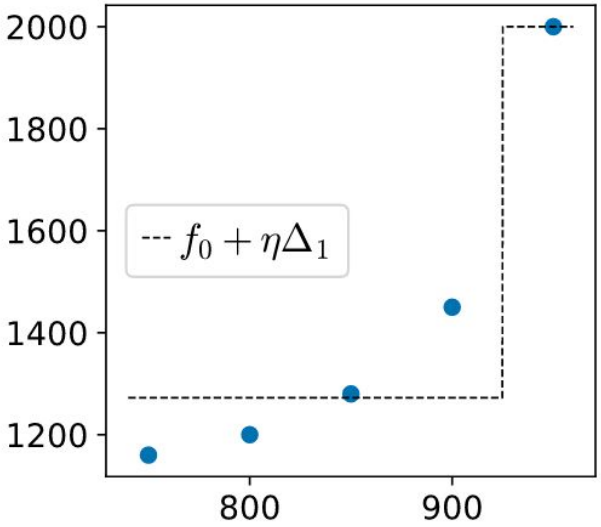
composite model sums together all of the weak models

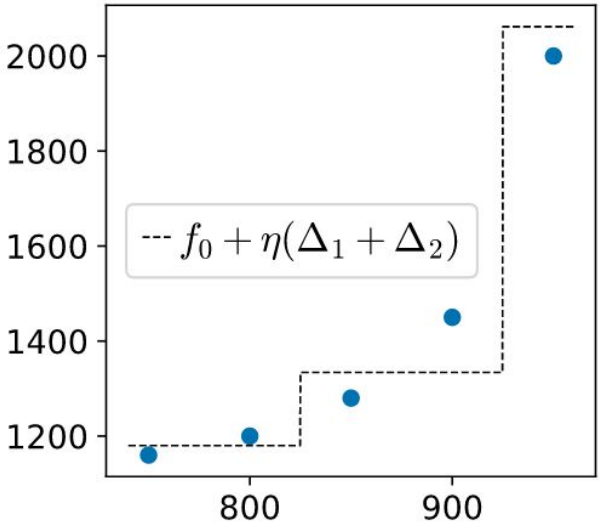


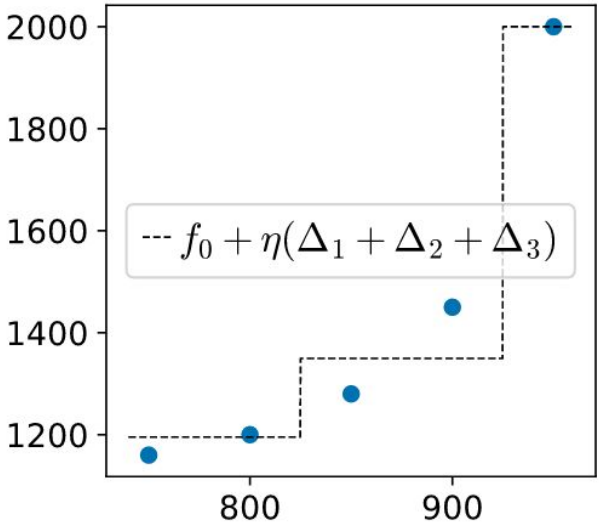
blue dots - residual vector elements used to train weak models,
dashed lines - predictions made by weak models,
dotted line - origin at 0

Example

adding all of weak models to the initial average model







Measuring model performance

The loss across all ***N*** observations is just the average of all the individual observation losses:

$$L(\mathbf{y}, F_M(X)) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, F_M(\mathbf{x}_i))$$

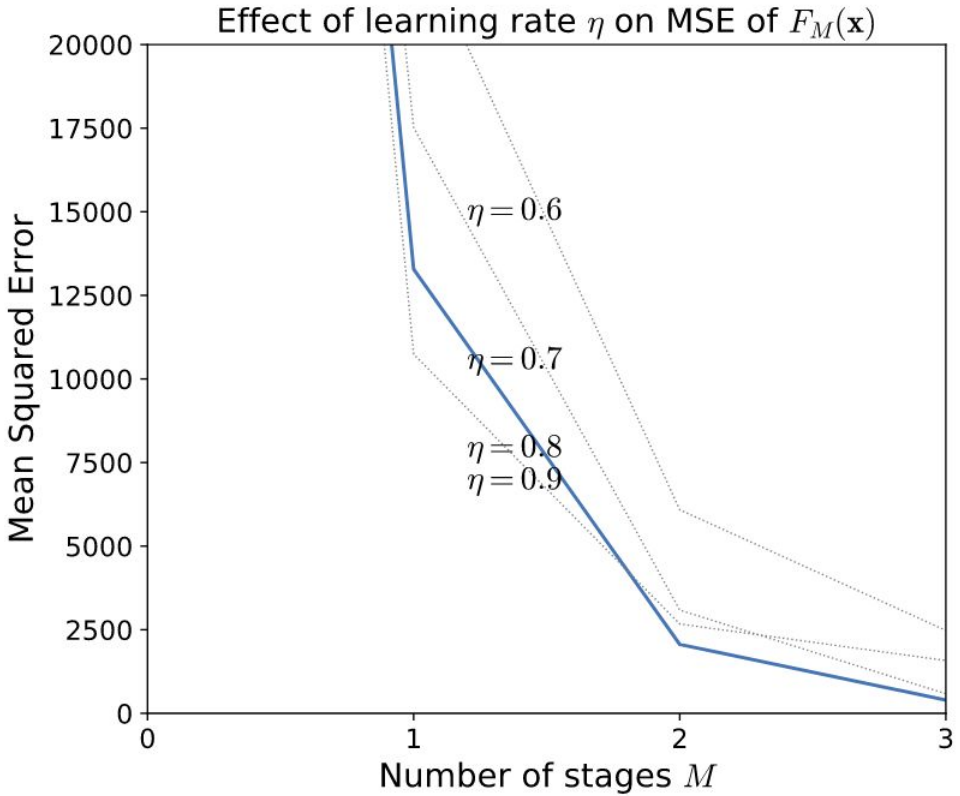
The mean squared error (MSE) is the most common, and what we are optimizing:

$$L(\mathbf{y}, F_M(X)) = \frac{1}{N}\sum_{i=1}^{N} (y_i - F_M(\mathbf{x}_i))^2$$

Choosing hyper-parameters

number of stages ***M***

the learning rate η



L2 loss optimizing algorithm

Algorithm: *l2boost*$(X,\mathbf{y},M,\eta)$ **returns** model F_M

Let $F_0(X) = \frac{1}{N}\sum_{i=1}^{N} y_i$, mean of target $\mathbf{y}$ across all observations

for $m = 1$ **to** M **do**

  Let $\mathbf{r}_{m-1} = \mathbf{y} - F_{m-1}(X)$ be the residual direction vector

  Train regression tree Δ_m on $\mathbf{r}_{m-1}$, minimizing squared error

  $F_m(X) = F_{m-1}(X) + \eta\Delta_m(X)$

end

return F_M

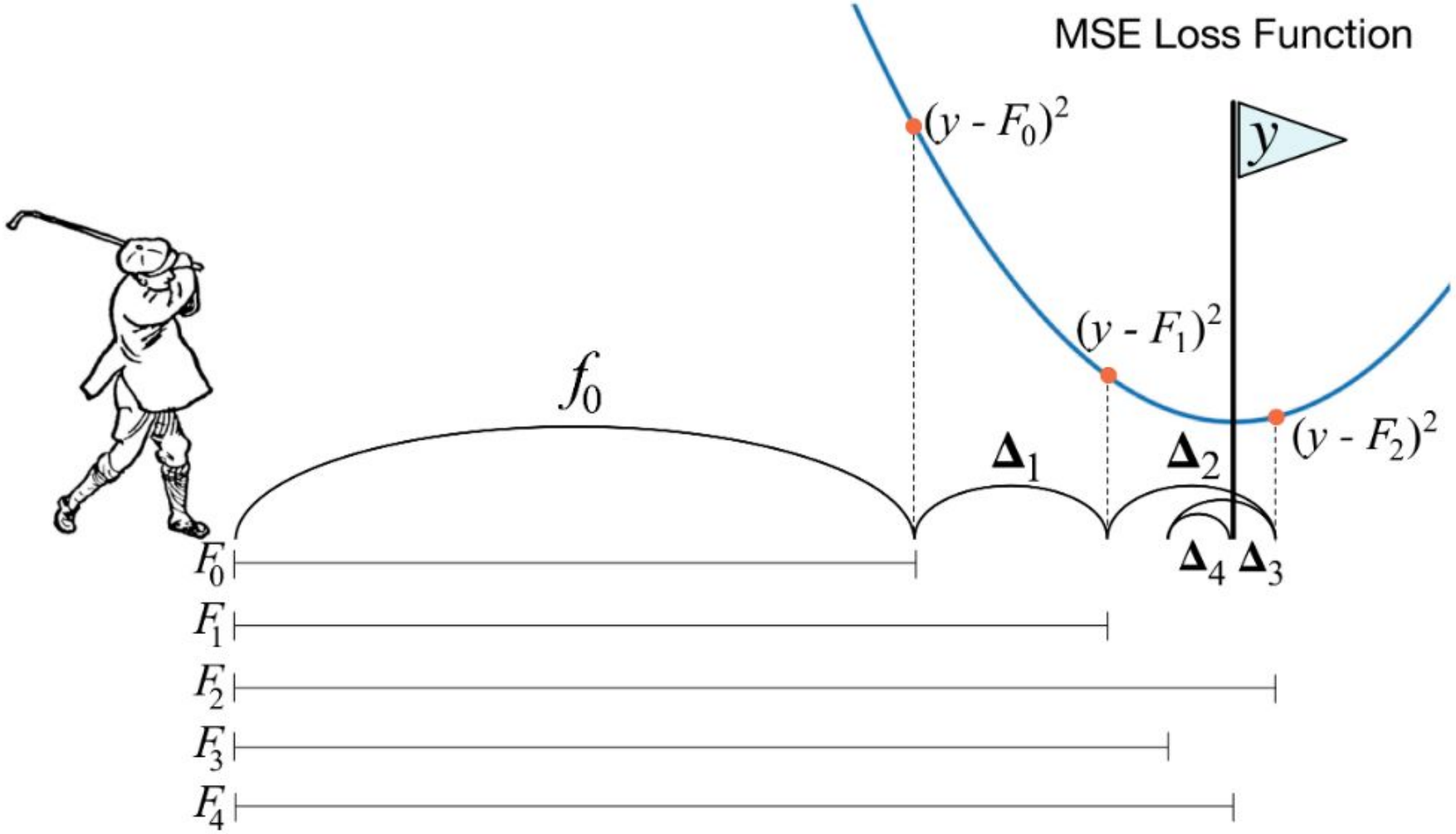
Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." *Annals of statistics* (2001): 1189-1232.

Why **gradient** boosting?

a mathematician's favorite trick: showing how our current problem is just a flavor of another well-known problem for which we have lots of useful results

GBM training weak learners on residual vectors optimizes the mean squared error (MSE), the L_2 loss, between the true target y and the intermediate predictions $F_m(X)$

Is adding weak models Δ_m ***to our GBM additive model*** $F_m(X) = F_{m-1}(X) + \eta\Delta_m(X)$ ***performing gradient descent in some way?***


MSE Loss Function
$(y - F_0)^2$
$(y - F_1)^2$
$(y - F_2)^2$
y
f_0
Δ_1
Δ_2
Δ_3
Δ_4
F_0
F_1
F_2
F_3
F_4

To uncover the loss function optimized we just have to integrate the residuals

$y - F_m(X)$

$$L(\mathbf{y}, F_M(X)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - F_M(\mathbf{x}_i))^2 \qquad L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

since ***N*** is a constant once we start boosting, and ***f(x)*** and ***cf(x)*** have the same ***x*** minimum point, let's drop the ***1/N*** constant

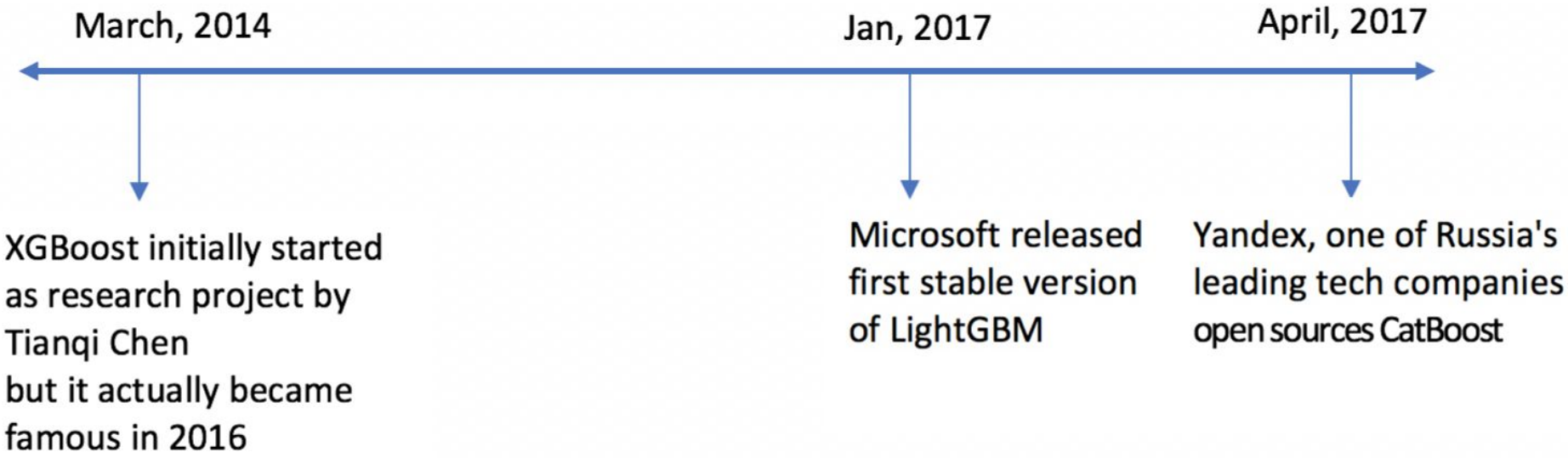
$$L(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

taking the partial derivative of the loss function with respect to a specific approximation $\hat{y}_j$

$$\begin{aligned}
\frac{\partial}{\partial \hat{y}_j} L(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{\partial}{\partial \hat{y}_j} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \\
&= \frac{\partial}{\partial \hat{y}_j} (y_j - \hat{y}_j)^2 \\
&= 2(y_j - \hat{y}_j) \frac{\partial}{\partial \hat{y}_j} (y_j - \hat{y}_j) \\
&= -2(y_j - \hat{y}_j)
\end{aligned}$$

remove the summation because the partial derivative of ***L*** for ***i≠j*** is 0

$$\nabla_{\hat{\mathbf{y}}} L(\mathbf{y}, \hat{\mathbf{y}}) = -2(\mathbf{y} - \hat{\mathbf{y}})$$


March, 2014
Jan, 2017
April, 2017
XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016
Microsoft released first stable version of LightGBM
Yandex, one of Russia's leading tech companies open sources CatBoost

XGBoost

Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016.

Difference between GBM and XGBoost?

- XGBoost and GBM follows the principle of gradient boosting
- XGBoost uses a more regularized model formalization to control complexity of the model and over-fitting, which gives it better performance.
- Objective Function : Training Loss + Regularization
- Regularization: complexity of the model

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training Loss measures how well model fit on training data

Regularization, measures complexity of trees

Complexity of a tree

Taken into account:

- Number of leaves in tree ***T***
- Leaf weigh penalty parameter γ
- Tree size penalty parameter λ
- Score of a leaf w_j

$$\Omega(F_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$Obj(F_t) = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

$$G_j = \sum_{i \in I_j} \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$H_j = \sum_{i \in I_j} \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$I = \{i | q(x_i) = j\}$$

LightGBM

LightGBM

Gradient-based One-Side Sampling (GOSS):

- exclude a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain
- since the data instances with larger gradients play a more important role in the computation of information gain, GOSS can obtain quite accurate estimation of the information gain with a much smaller data size -> FASTER

Exclusive Feature Bundling (EFB):

- bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features
- finding the optimal bundling of exclusive features is NP-hard, but a greedy algorithm can achieve quite good approximation ratio

Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." *Advances in Neural Information Processing Systems*. 2017.

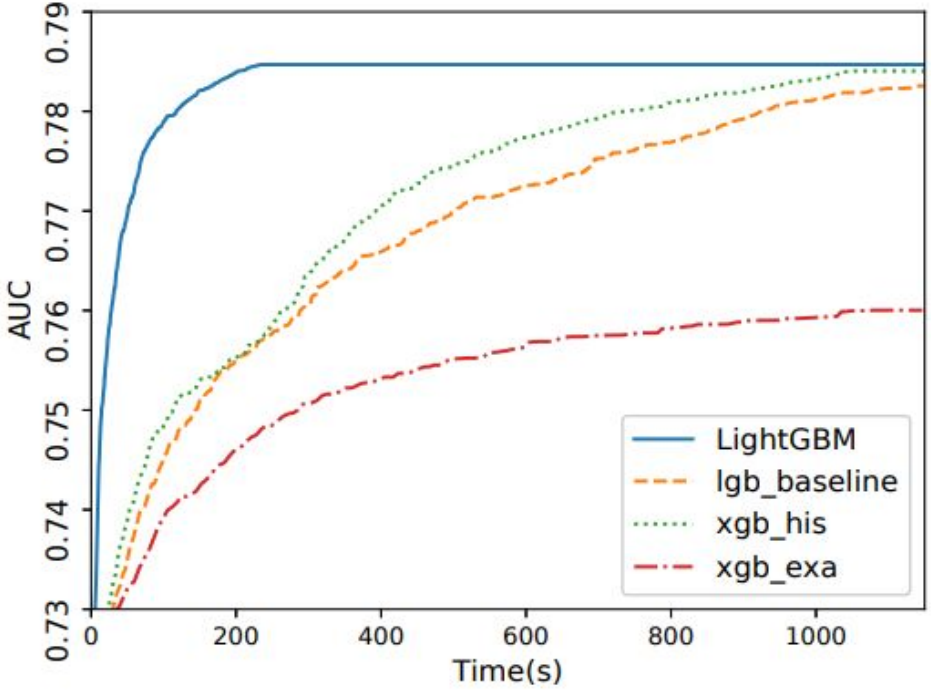


Figure 1: Time-AUC curve on Flight Delay.

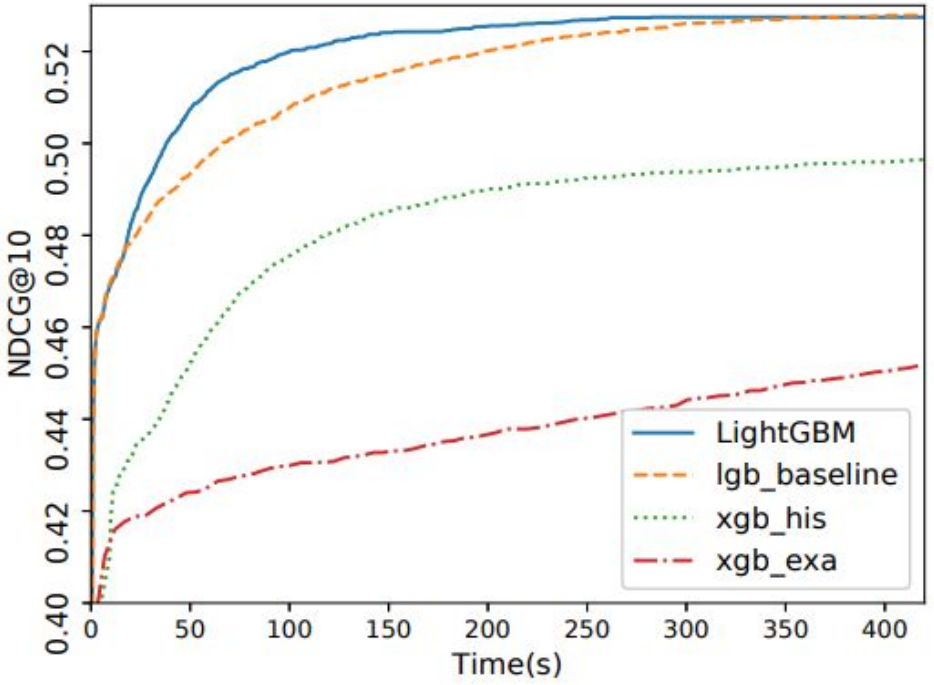


Figure 2: Time-NDCG curve on LETOR.

CatBoost

CatBoost

- ordered boosting - a permutation-driven alternative to the classic algorithm
  - Adresses the **prediction shift** problem - gradients used at each step are estimated using the target values of the same data points the current model F_{m-1} was built on
- an innovative algorithm for processing categorical features

Prokhorenkova, Liudmila, et al. "CatBoost: unbiased boosting with categorical features." *Advances in Neural Information Processing Systems*. 2018.

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

Conclusions