Machine Learning Course

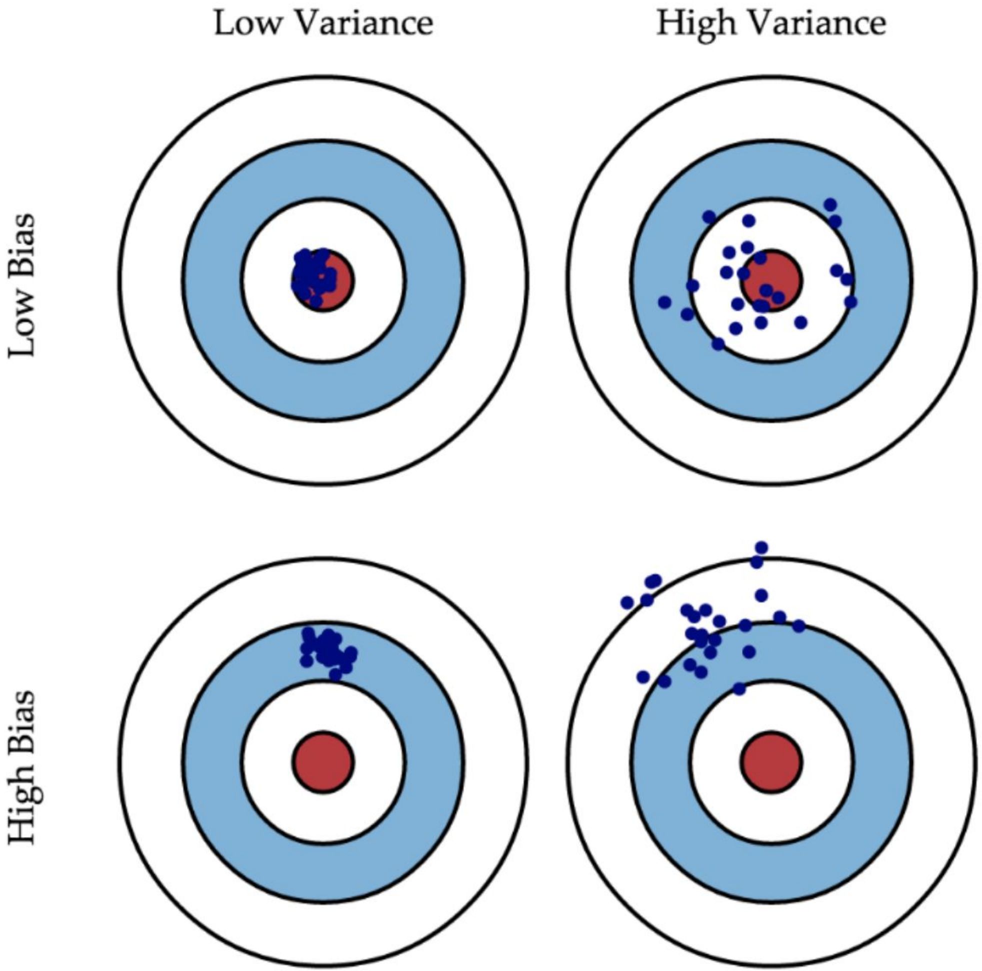
Lecture 7: Ensembles 2

MIPT, 2019

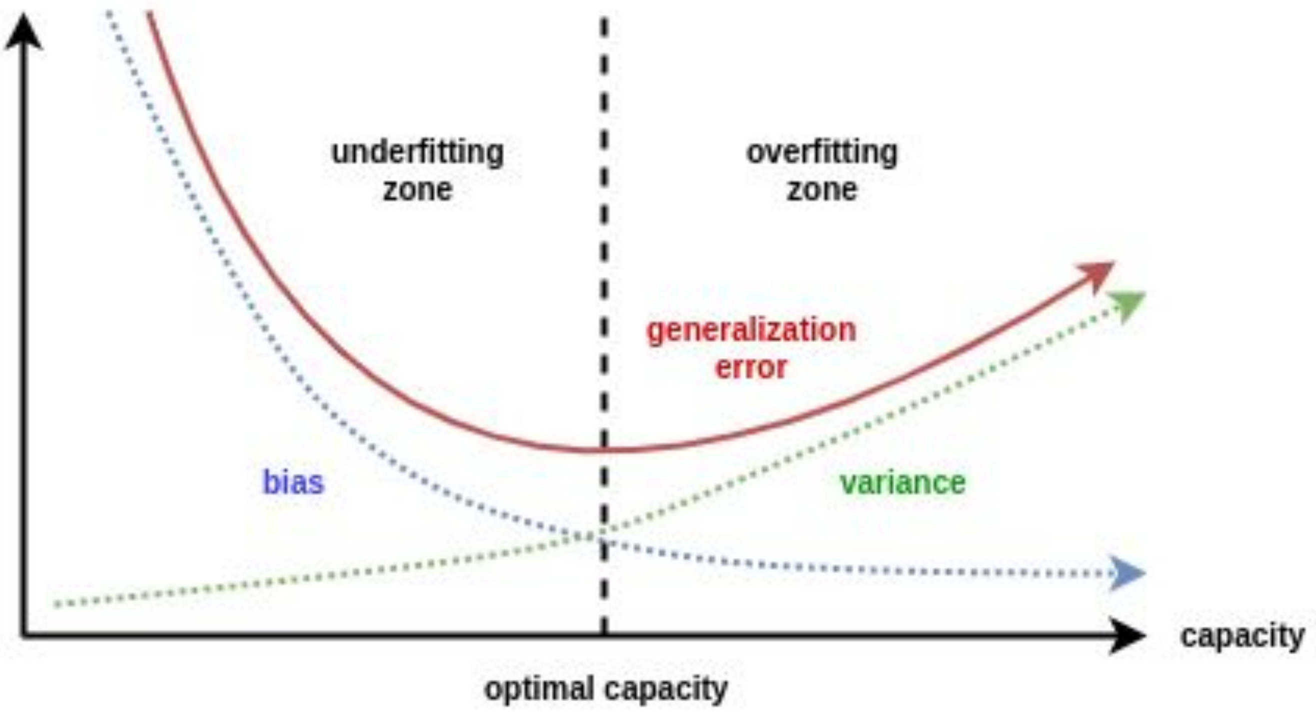
Outline

1. Bias-variance tradeoff recap.
2. Stacking.
3. Blending.
4. Gradient boosting.

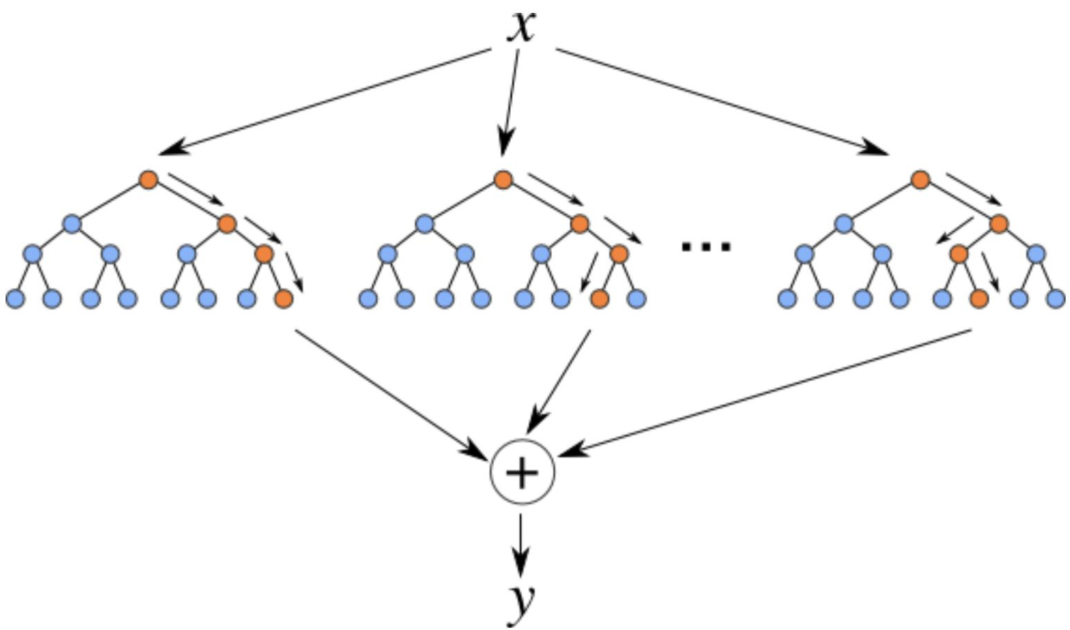
Based on following materials: ml-mipt 2017 by Victor Kantor, Evgeny Sokolov HSE 2018, mlcourse_open by ODS


Low Variance
High Variance
Low Bias
High Bias

Bias-variance tradeoff

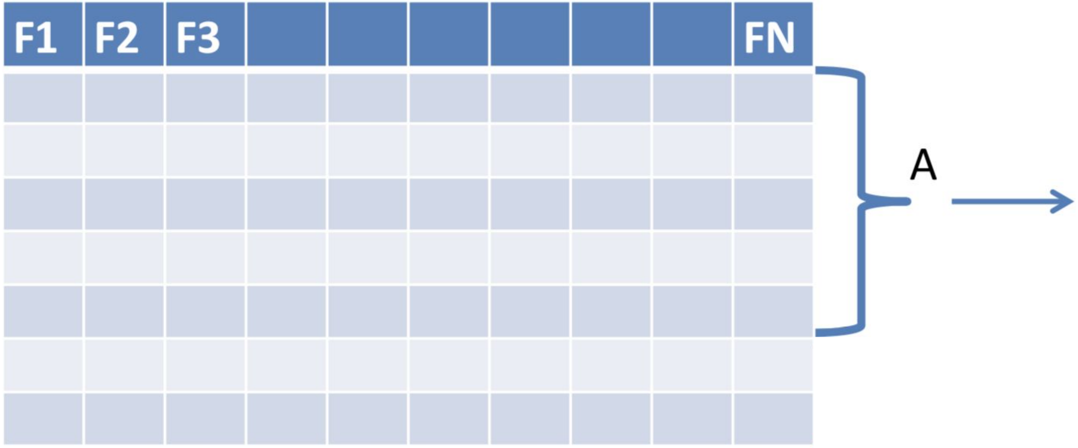


Bagging + RSM = Random Forest



Stacking

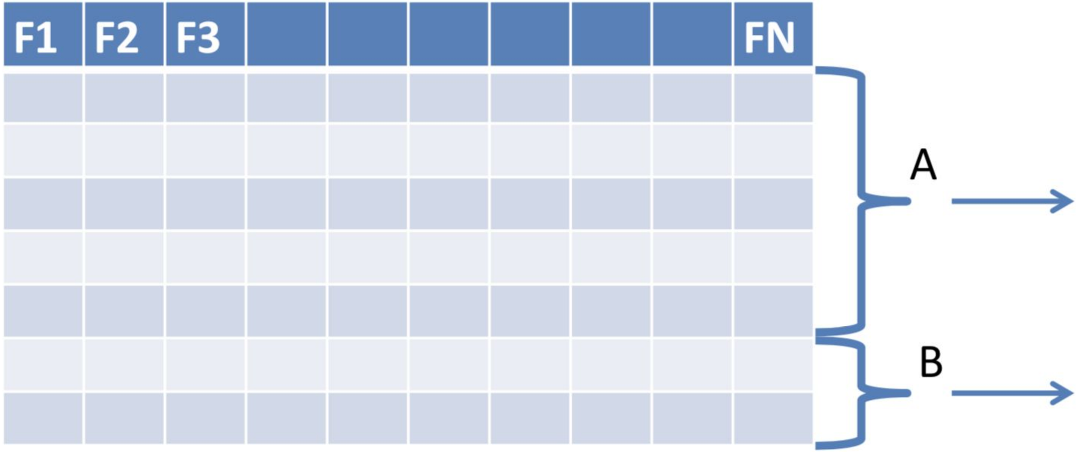
How to build an ensemble from *different* models?



Tune M basic algorithms

Stacking

How to build an ensemble from *different* models?

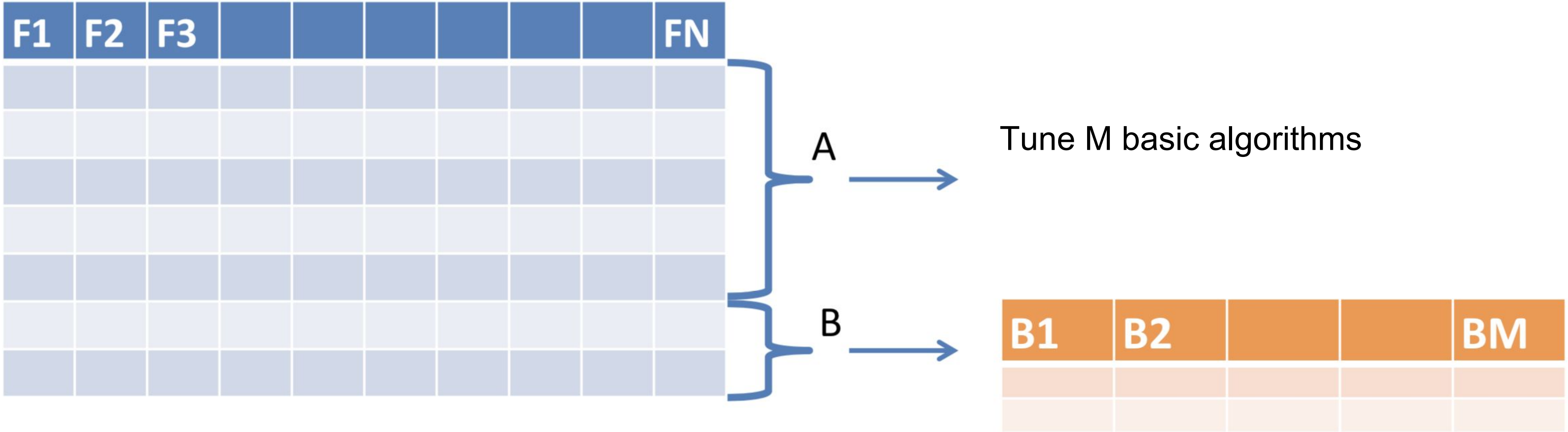


Tune M basic algorithms

Tune new model on the outputs of the basic algorithms

Stacking

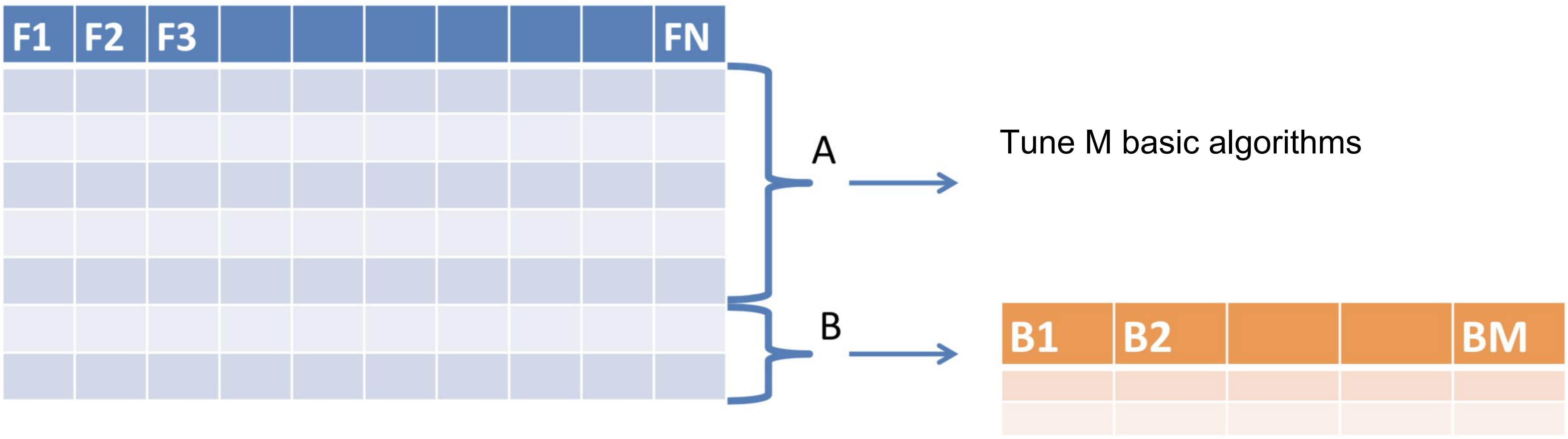
How to build an ensemble from *different* models?



Tune M basic algorithms

Stacking

How to build an ensemble from *different* models?



e.g.

$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x)$$

Stacking

How to build an ensemble from *different* models?

Stacking

How to build an ensemble from *different* models?

- Use different datasets (or datasets parts) for different level models.

Stacking

How to build an ensemble from *different* models?

- Use different datasets (or datasets parts) for different level models.
- Experiment with different models (linear, trees ensembles, simple networks, etc.)

Stacking

How to build an ensemble from *different* models?

- Use different datasets (or datasets parts) for different level models.
- Experiment with different models (linear, trees ensembles, simple networks, etc.)
- Or just different GBT ensembles (hola, kaggle :)

Blending

Just combine several *strong/complex* models.

Weights should sum up to 1
and come from [0; 1]

$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x)$$

Blending

Just combine several *strong/complex* models.

Weights should sum up to 1
and come from [0; 1]

$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x)$$

- Simple and intuitive ensembling method

Blending

Just combine several *strong/complex* models.

Weights should sum up to 1
and come from [0; 1]

$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x)$$

- Simple and intuitive ensembling method.
- Finding optimal weights could be tricky.

Blending

Just combine several *strong/complex* models.

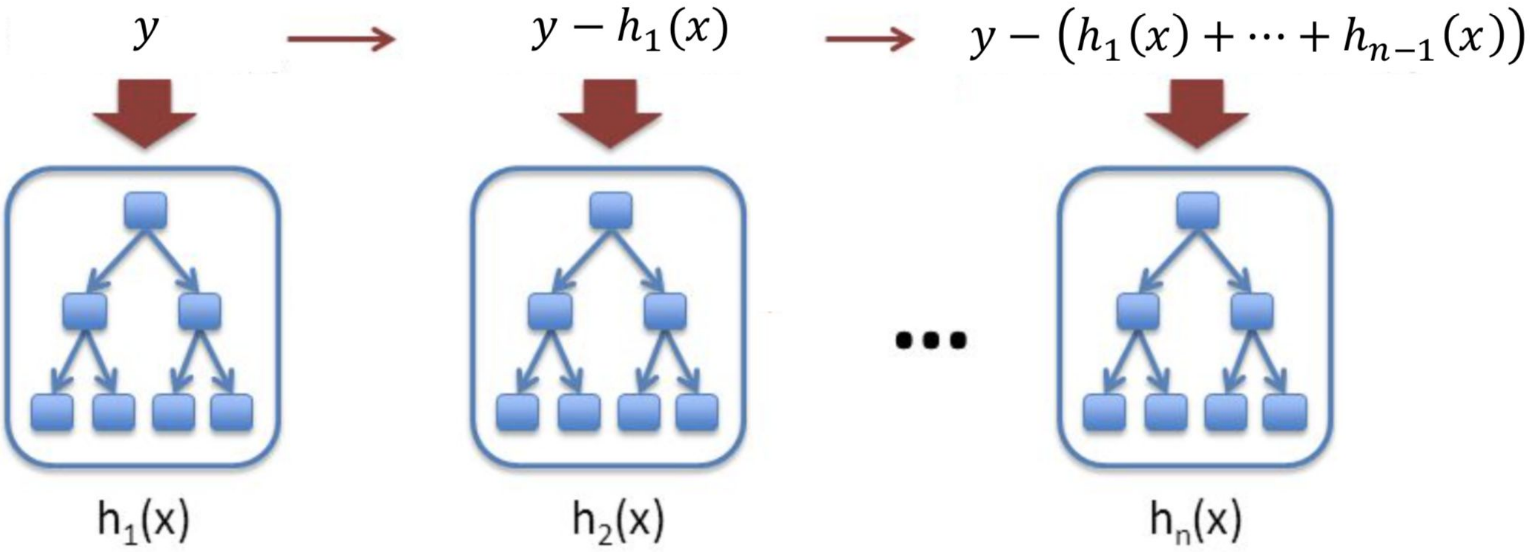
Weights should sum up to 1
and come from [0; 1]

$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x)$$

- Simple and intuitive ensembling method.
- Finding optimal weights could be tricky.
- Linear composition is not always enough.

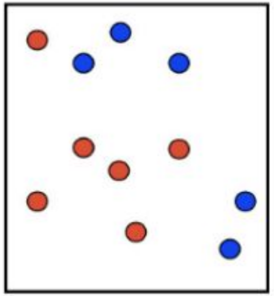
Gradient boosting

$$a_n(x) = h_1(x) + \cdots + h_n(x)$$

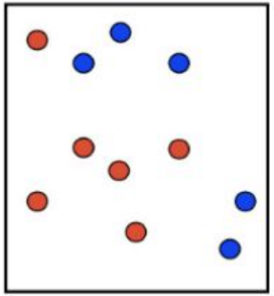


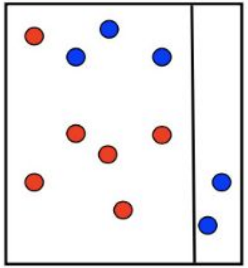
Boosting: intuition

Binary classification problem.
Models - decision stumps.

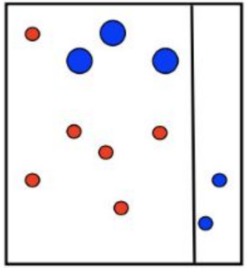


Boosting: intuition

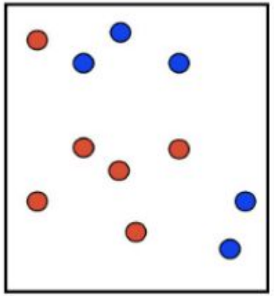


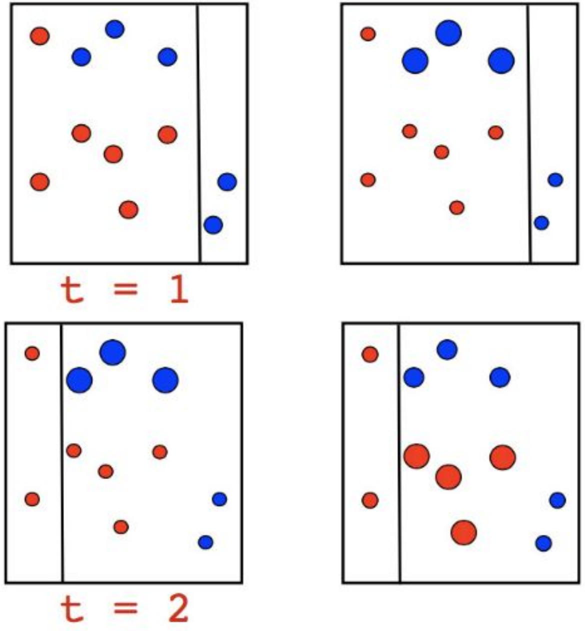


t = 1

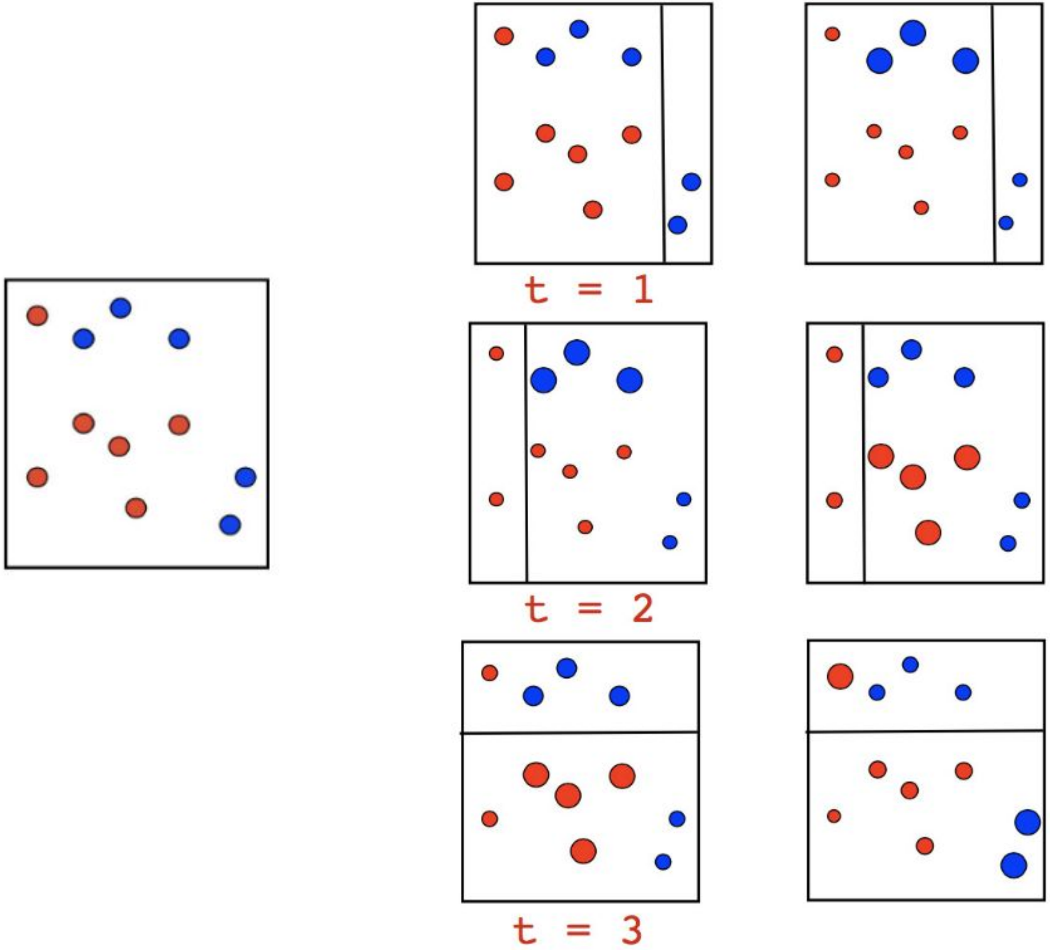


Boosting: intuition



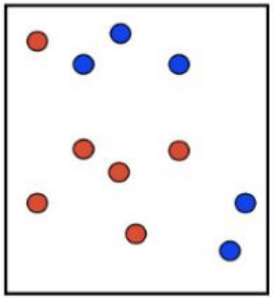


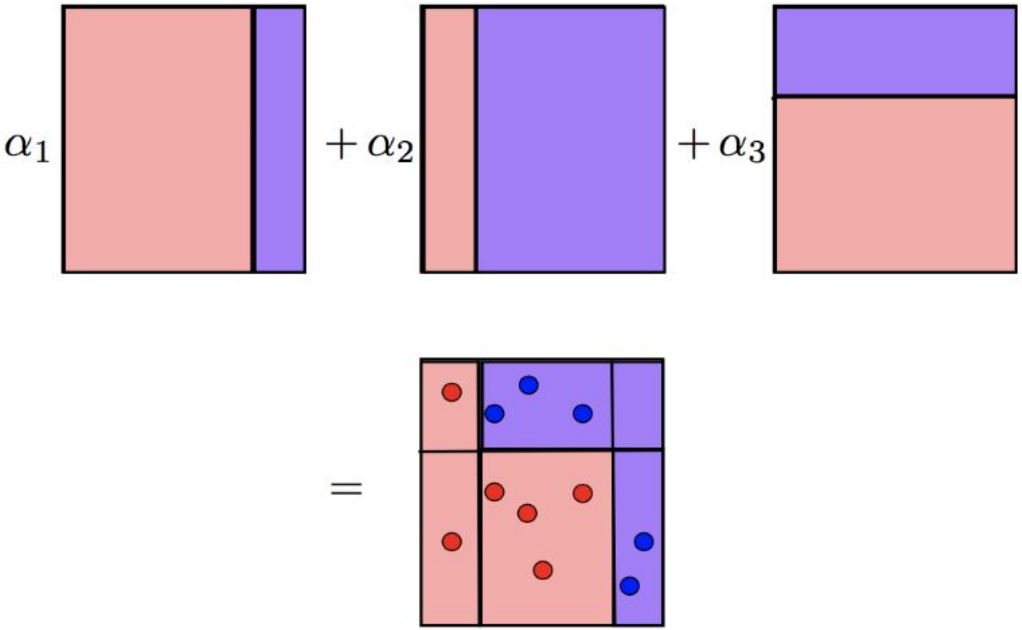
Boosting: intuition



Boosting: intuition

Binary classification problem.
Models - decision stumps.





Gradient boosting: theory

Denote dataset $\{(x_i, y_i)\}_{i=1,\dots,n}$, loss function $L(y, f)$.

Gradient boosting: theory

Denote dataset $\{(x_i, y_i)\}_{i=1,\dots,n}$, loss function $L(y, f)$.

Optimal model:

$$\hat{f}(x) = \underset{f(x)}{\arg\min}\ L(y, f(x)) = \underset{f(x)}{\arg\min}\ \mathbb{E}_{x,y}[L(y, f(x))]$$

Gradient boosting: theory

Denote dataset $\{(x_i, y_i)\}_{i=1,\dots,n}$, loss function $L(y, f)$.

Optimal model:

$$\hat{f}(x) = \underset{f(x)}{\arg\min}\ L(y, f(x)) = \underset{f(x)}{\arg\min}\ \mathbb{E}_{x,y}[L(y, f(x))]$$

Let it be from parametric family: $\hat{f}(x) = f(x, \hat{\theta})$,

$$\hat{\theta} = \underset{\theta}{\arg\min}\ \mathbb{E}_{x,y}[L(y, f(x, \theta))]$$

Gradient boosting: theory

$$\hat{f}(x) = \sum_{i=0}^{t-1} \hat{f}_i(x),$$

$$(\rho_t, \theta_t) = \arg\min_{\rho,\theta} \mathbb{E}_{x,y}[L(y, \hat{f}(x) + \rho \cdot h(x, \theta))],$$

$$\hat{f}_t(x) = \rho_t \cdot h(x, \theta_t)$$

Gradient boosting: theory

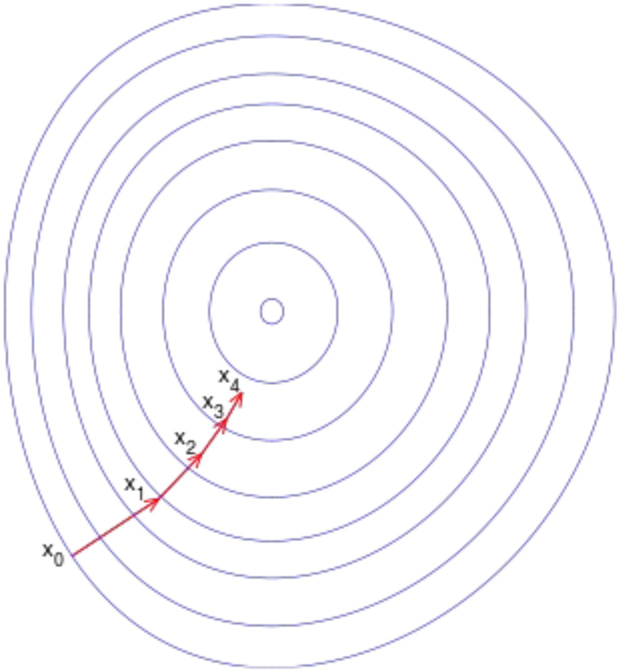
$$\hat{f}(x) = \sum_{i=0}^{t-1} \hat{f}_i(x),$$

$$(\rho_t, \theta_t) = \arg\min_{\rho,\theta} \mathbb{E}_{x,y}[L(y, \hat{f}(x) + \rho \cdot h(x,\theta))],$$

$$\hat{f}_t(x) = \rho_t \cdot h(x, \theta_t)$$

What if we could use gradient descent in *space of our models*?

Gradient boosting: theory



What if we could use gradient descent in *space of our models*?

Gradient boosting: theory

$$\hat{f}(x) = \sum_{i=0}^{t-1} \hat{f}_i(x),$$

$$r_{it} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}(x)}, \quad \text{for } i = 1, \ldots, n,$$

$$\theta_t = \arg\min_{\theta} \sum_{i=1}^{n} (r_{it} - h(x_i, \theta))^2,$$

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, \hat{f}(x_i) + \rho \cdot h(x_i, \theta_t))$$

Gradient boosting: theory

In linear regression case with MSE loss:

$$r_{it} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}(x)} = -2(\hat{y}_i - y_i) \propto \hat{y}_i - y_i$$

Gradient boosting: theory

What we need:

- Data.
- Loss function and its gradient.
- Family of algorithms (with constraints on hyperparameters if necessary).
- Number of iterations M.
- Initial value (GBM by Friedman): constant.

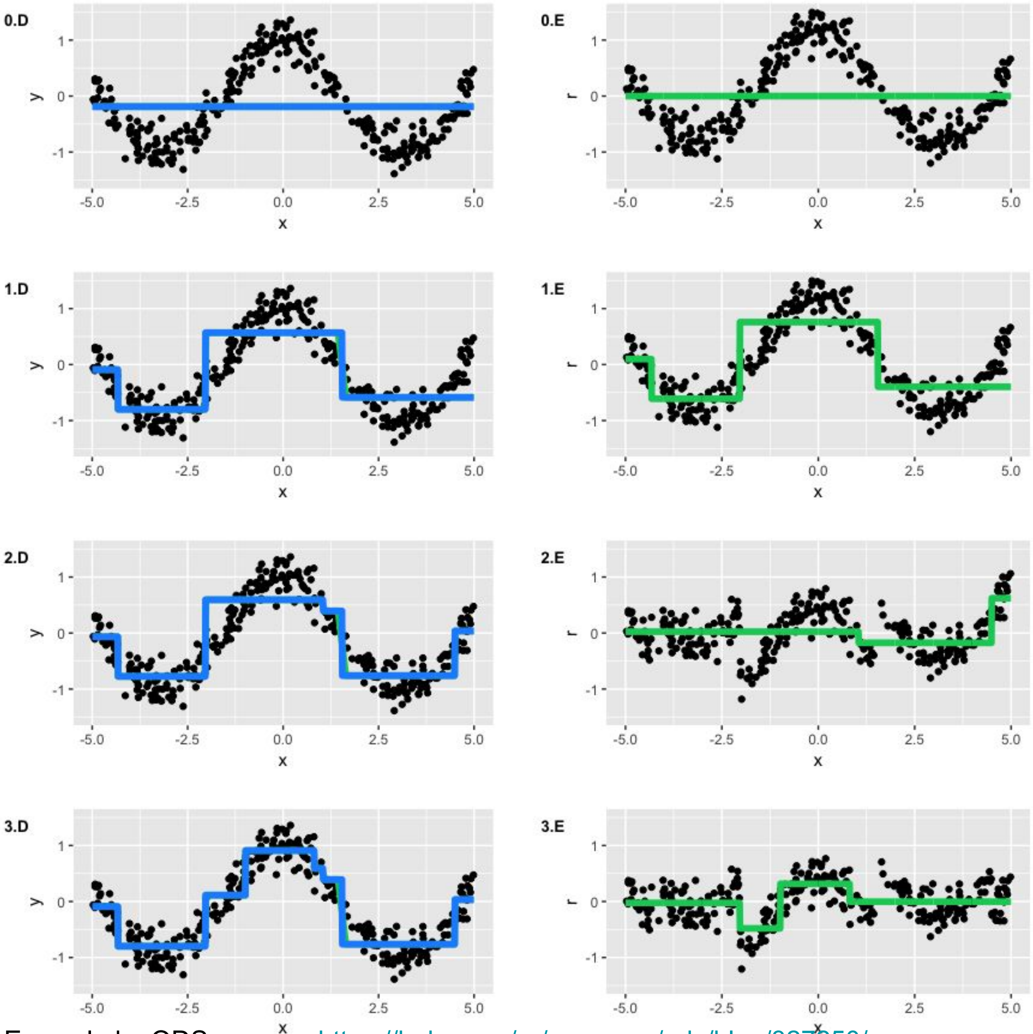
Gradient boosting: example

What we need:

- Data: toy dataset $y = cos(x) + \epsilon, \epsilon \sim \mathcal{N}(0, \frac{1}{5}), x \in [-5, 5]$
- Loss function: MSE
- Family of algorithms: decision trees with depth 2
- Number of iterations M = 3
- Initial value: just mean value

Example by ODS; source: https://habr.com/ru/company/ods/blog/327250/

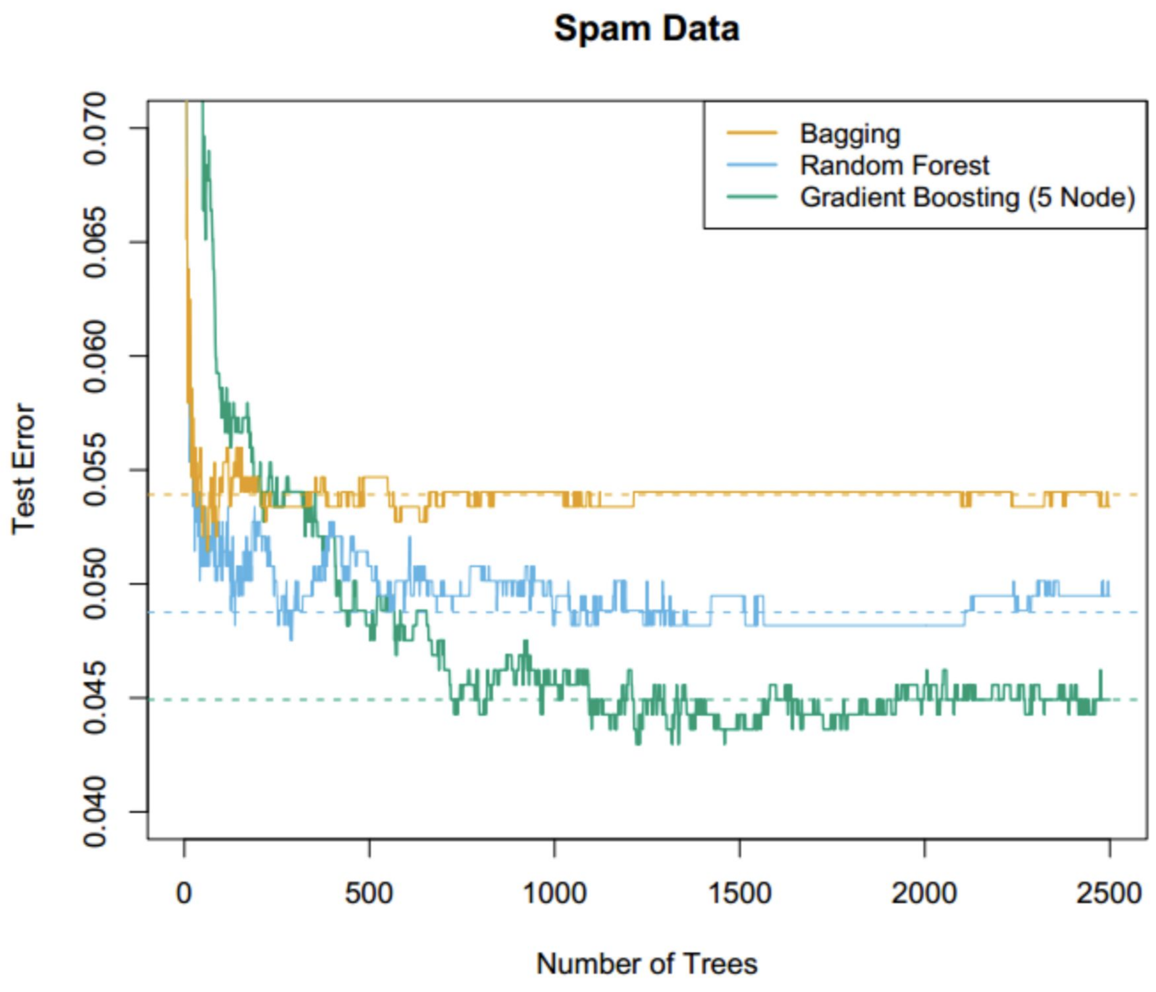
Gradient boosting: example



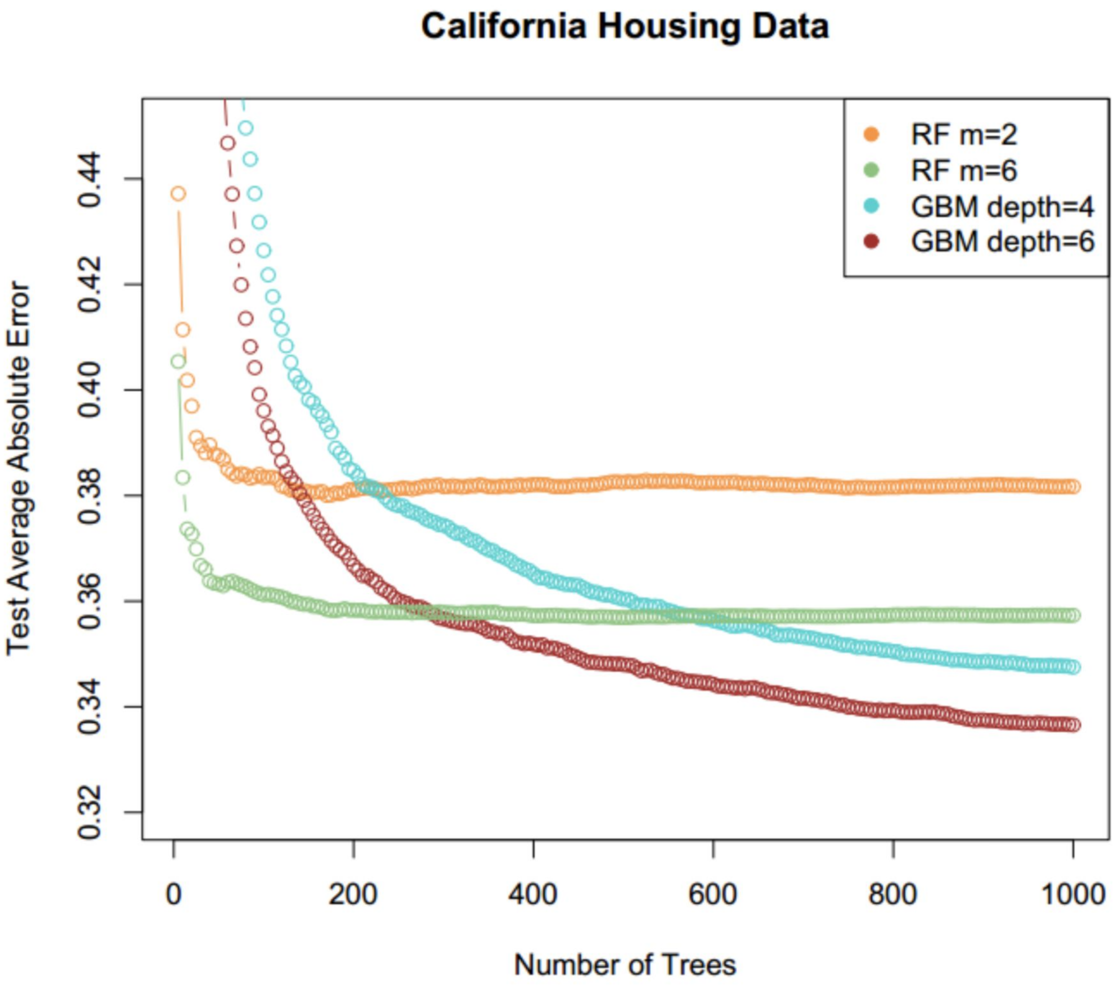
Left: full ensemble on each step.

Right: additional tree decisions.

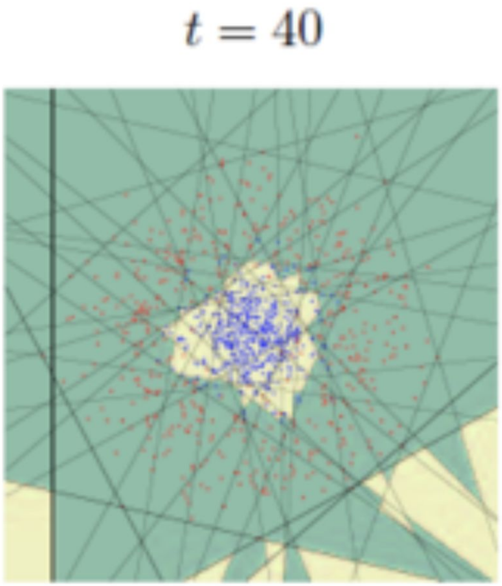
Example by ODS; source: https://habr.com/ru/company/ods/blog/327250/

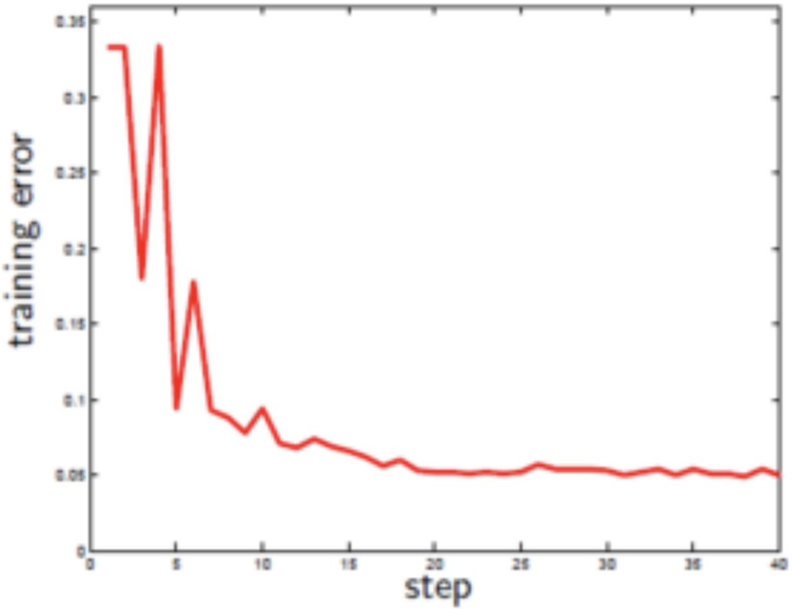


Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
0
500
1000
1500
2000
2500
Number of Trees


California Housing Data
RF m=2
RF m=6
GBM depth=4
GBM depth=6
Test Average Absolute Error
0.32
0.34
0.36
0.38
0.40
0.42
0.44
0
200
400
600
800
1000
Number of Trees

Boosting with linear classification methods





Technical side: training in parallel

Which of the ensembling methods could be parallelized?

Technical side: training in parallel

Which of the ensembling methods could be parallelized?

- Random Forest: parallel on the forest level (all trees are independent)

Technical side: training in parallel

Which of the ensembling methods could be parallelized?

- Random Forest: parallel on the forest level (all trees are independent)
- Gradient boosting: parallel on one tree level

Recap: ensembling methods

1. Bagging.
2. Random subspace method (RSM).
3. Bagging + RSM + Decision trees = Random Forest.
4. Gradient boosting.
5. Stacking.
6. Blending.

Great demo: http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

Extra lecture about feature engineering and ML techniques is coming next week. Stay tuned.