MADMO

Boostings and ensembles. Part 2.

Taras Khakhulin
Deep Learning Engineer Samsung AI Center
Skoltech & MIPT MS

t.khakhulin@gmail.com
https://github.com/khakhulin/
https://twitter.com/t_khakhulin

Ensemble problems

Fixed target function

Similar dataset

Solve one task

Ensembles

Voting

- averaging

Stacking

Boosting

Output Coding

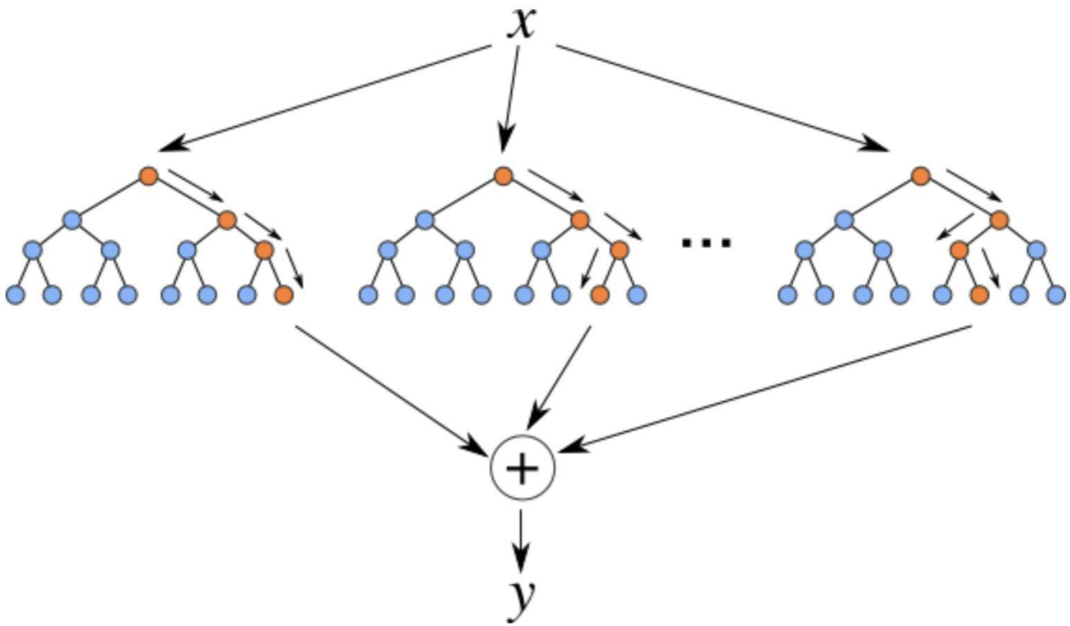
- code target (squared)

Bagging

Heuristics

- Hand-crafted methods

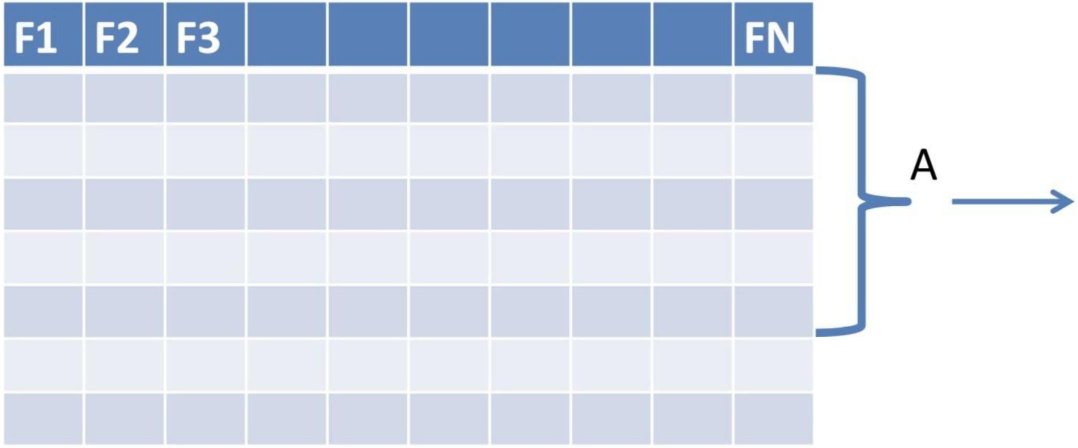
Bagging + RSM = Random Forest



Lecture 1. Part 2

Stacking

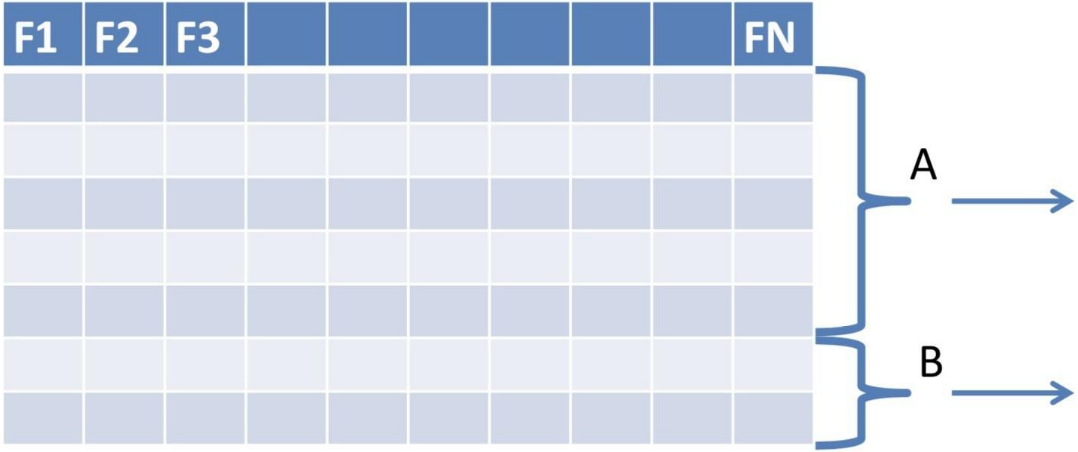
How to build an ensemble from *different* models?



Tune M basic algorithms

Stacking

How to build an ensemble from *different* models?

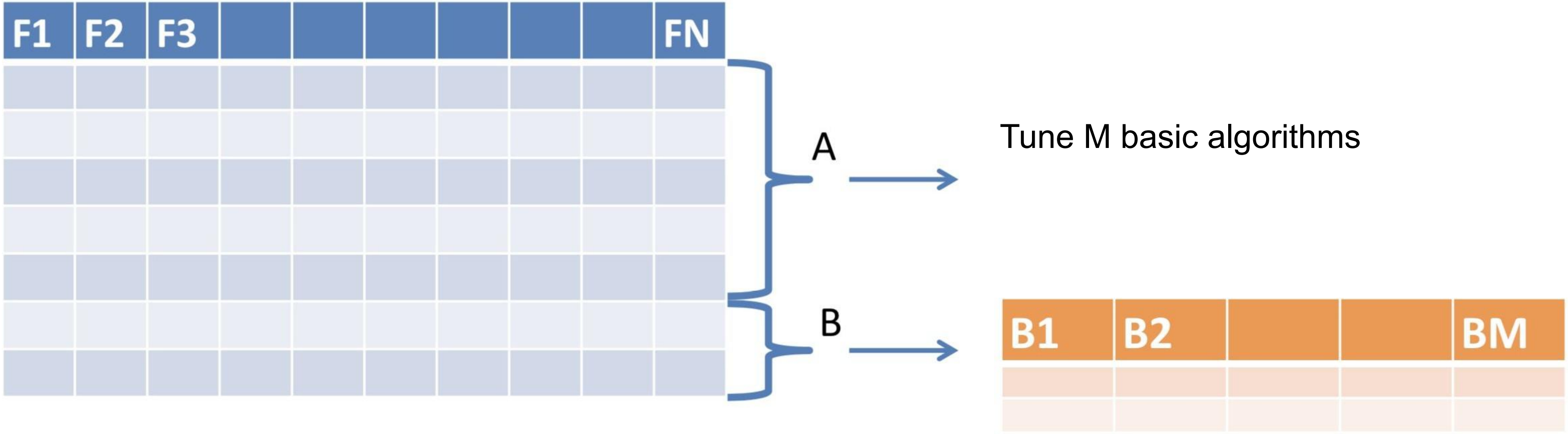


Tune M basic algorithms

Tune new model on the outputs of the basic algorithms

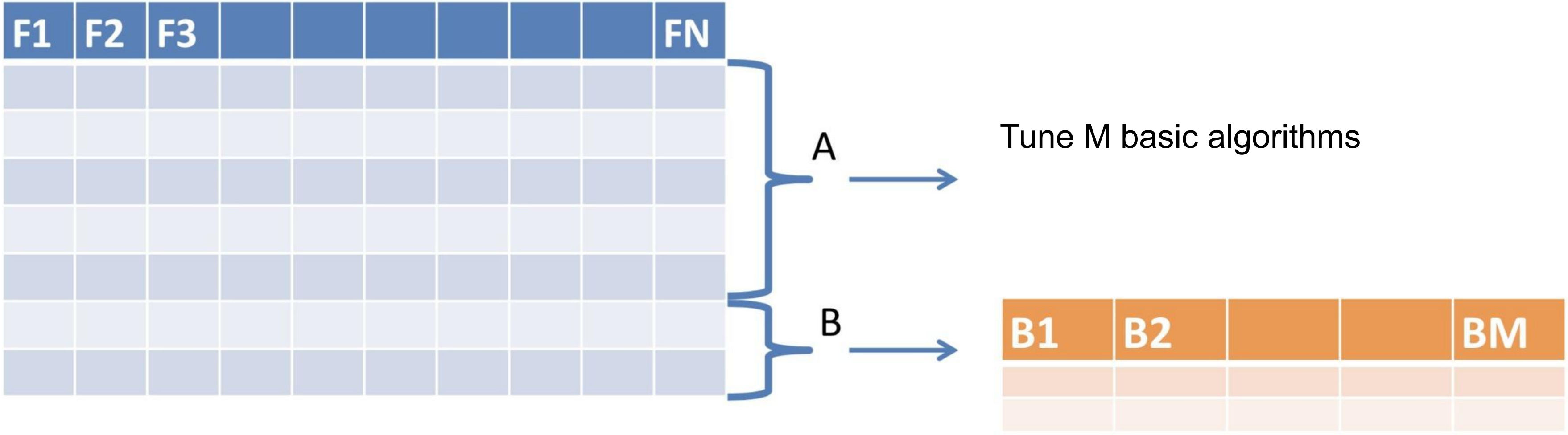
Stacking

How to build an ensemble from *different* models?



Stacking

How to build an ensemble from *different* models?



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

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.

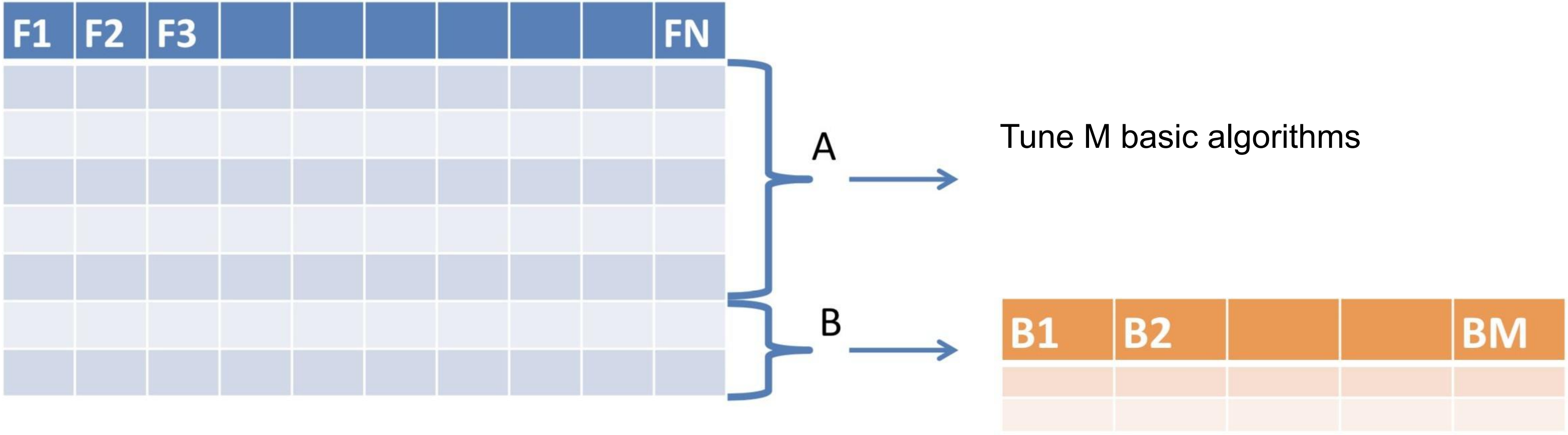
Weights should sum up to 1
and come from [0; 1]

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

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

Blending as a Stacking

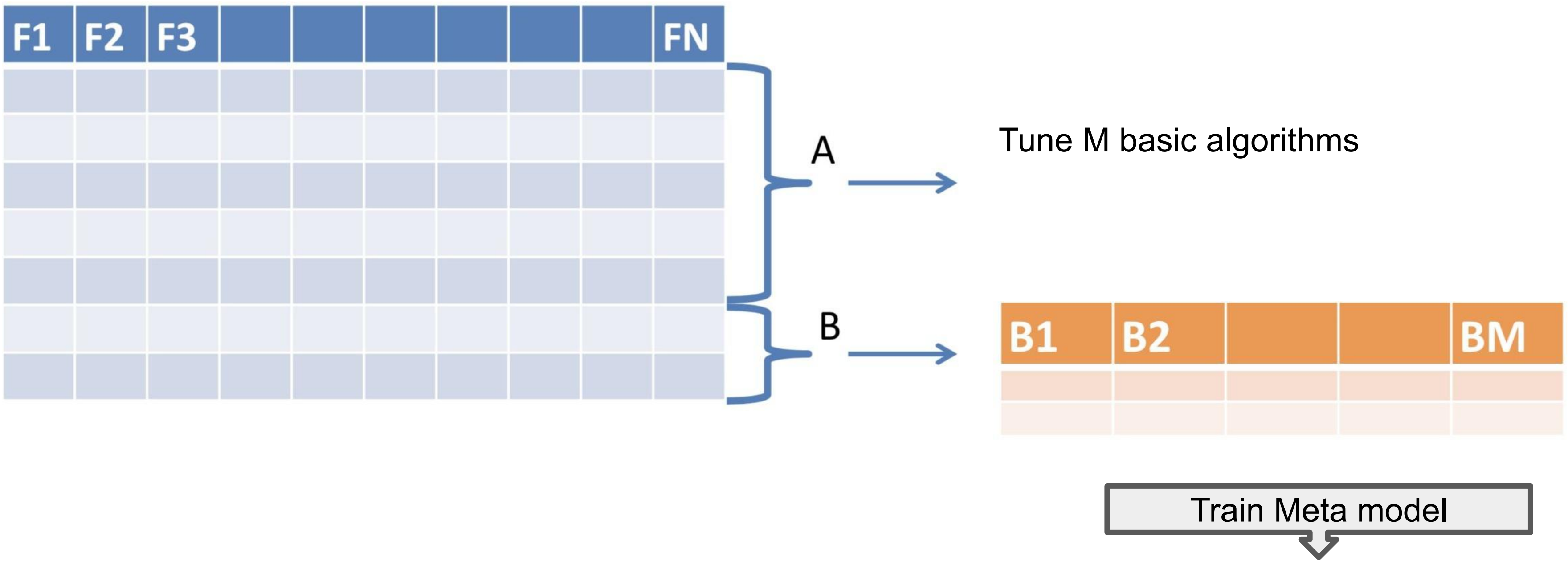
How to build an ensemble from *different* models?



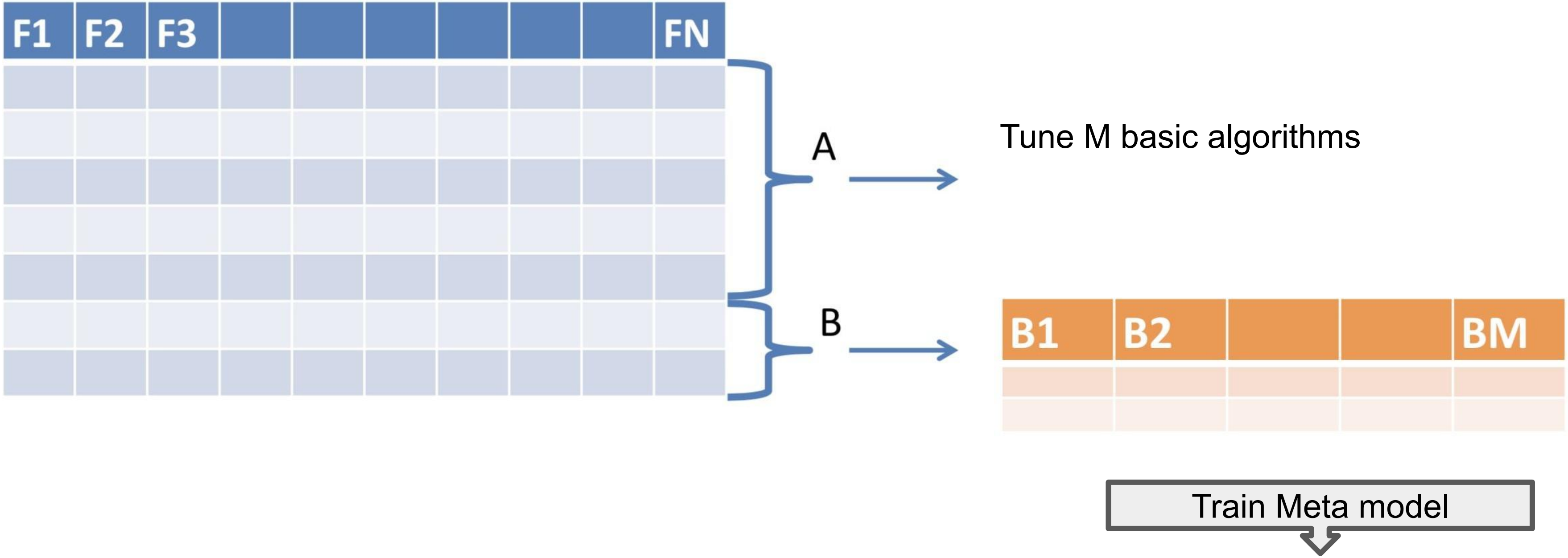
$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x)$$

e.g.

Blending as a Stacking

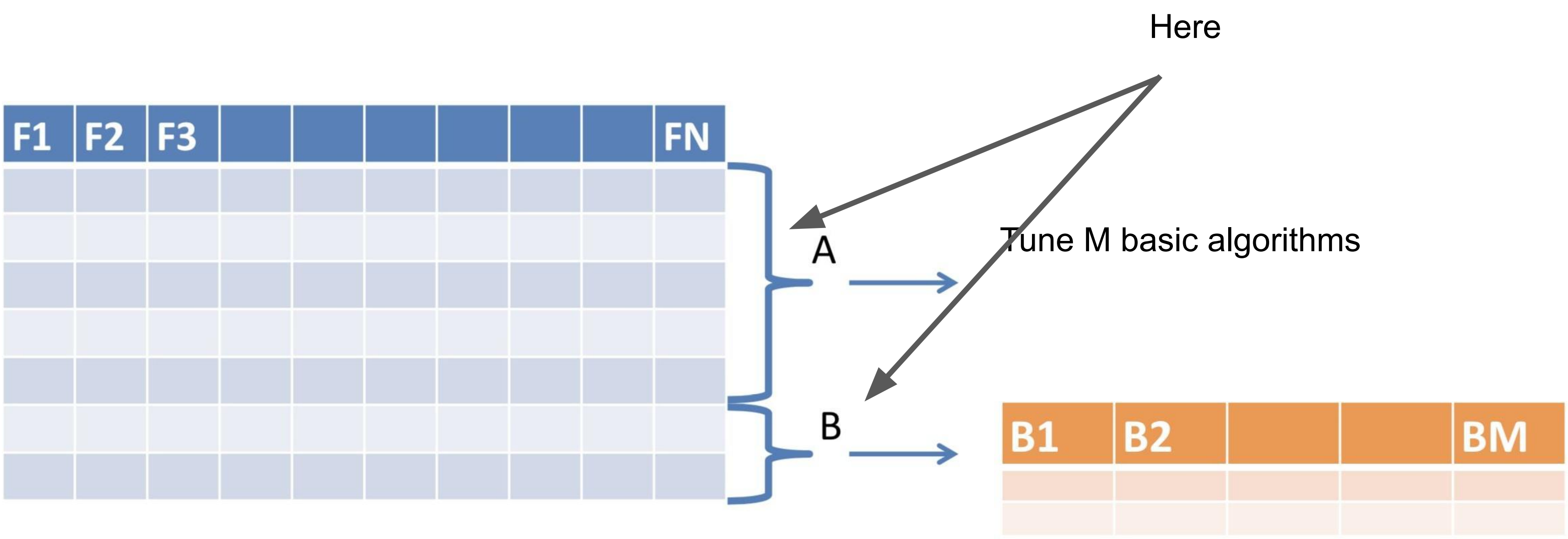


Blending as a Stacking

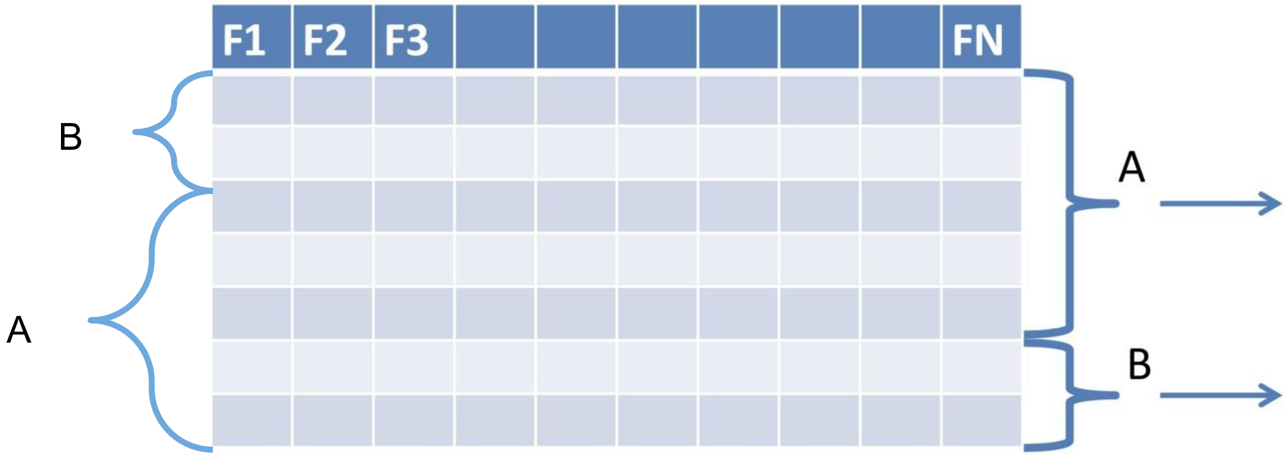


Where is a problem?

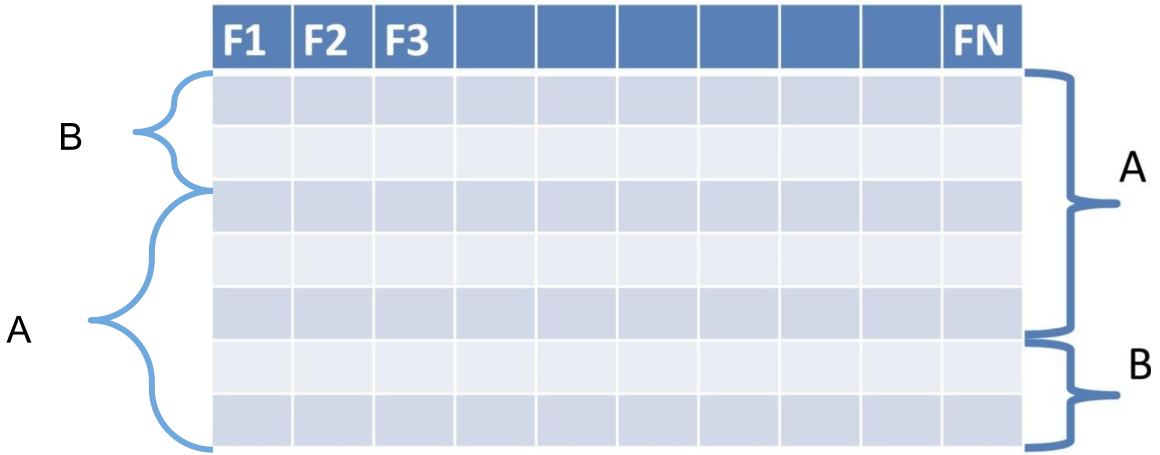
Blending as a Stacking



Blending-2



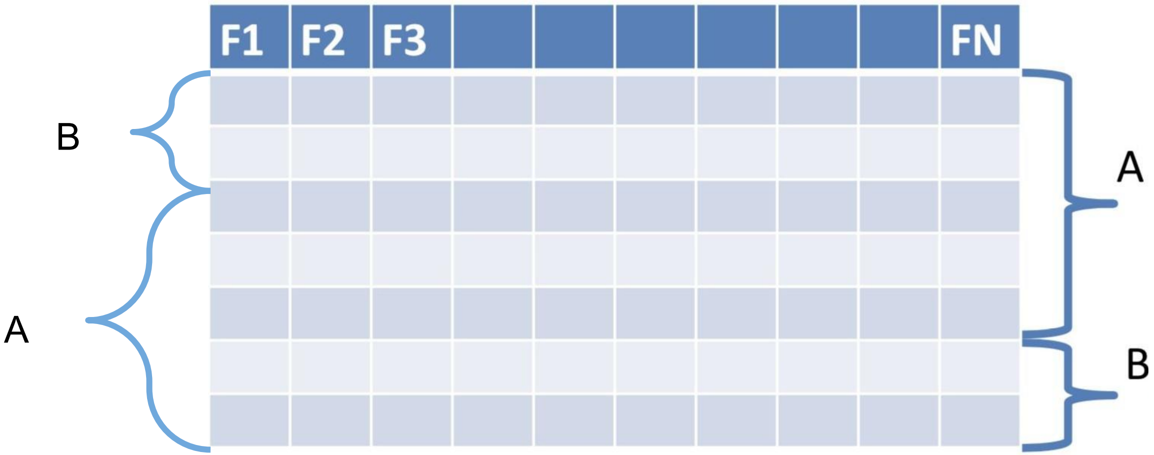
Blending-2



2nd meta model

1st meta model

Blending-2



2nd meta model

Ensemble them

1st meta model

Still confused about Stacking?

A				
X0	x1	x2	xn	y
0.17	0.25	0.93	0.79	1
0.35	0.61	0.93	0.57	0
0.44	0.59	0.56	0.46	0
0.37	0.43	0.74	0.28	1
0.96	0.07	0.57	0.01	1

B				
X0	x1	x2	xn	y
0.89	0.72	0.50	0.66	0
0.58	0.71	0.92	0.27	1
0.10	0.35	0.27	0.37	0
0.47	0.68	0.30	0.98	0
0.39	0.53	0.59	0.18	1

C				
X0	x1	x2	xn	y
0.29	0.77	0.05	0.09	?
0.38	0.66	0.42	0.91	?
0.72	0.66	0.92	0.11	?
0.70	0.37	0.91	0.17	?
0.59	0.98	0.93	0.65	?

Train algorithm **0** on A and make predictions for B and C and save to **B1**, **C1**

Train algorithm **1** on A and make predictions for B and C and save to **B1**, **C1**

Train algorithm **2** on A and make predictions for B and C and save to **B1**, **C1**

B1			
pred0	pred1	pred2	y
0.24	0.72	0.70	0
0.95	0.25	0.22	1
0.64	0.80	0.96	0
0.89	0.58	0.52	0
0.11	0.20	0.93	1

C1				
pred0	pred1	pred2	y	Preds3
0.50	0.50	0.39	?	0.45
0.62	0.59	0.46	?	0.23
0.22	0.31	0.54	?	0.99
0.90	0.47	0.09	?	0.34
0.20	0.09	0.61	?	0.05

Train algorithm **3** on B1 and make predictions for C1

Stacking

- Correlated models
- Use models with different natures
- Understand your models
- Work with your feature space
- Stack over stacking StackNet

Stacking

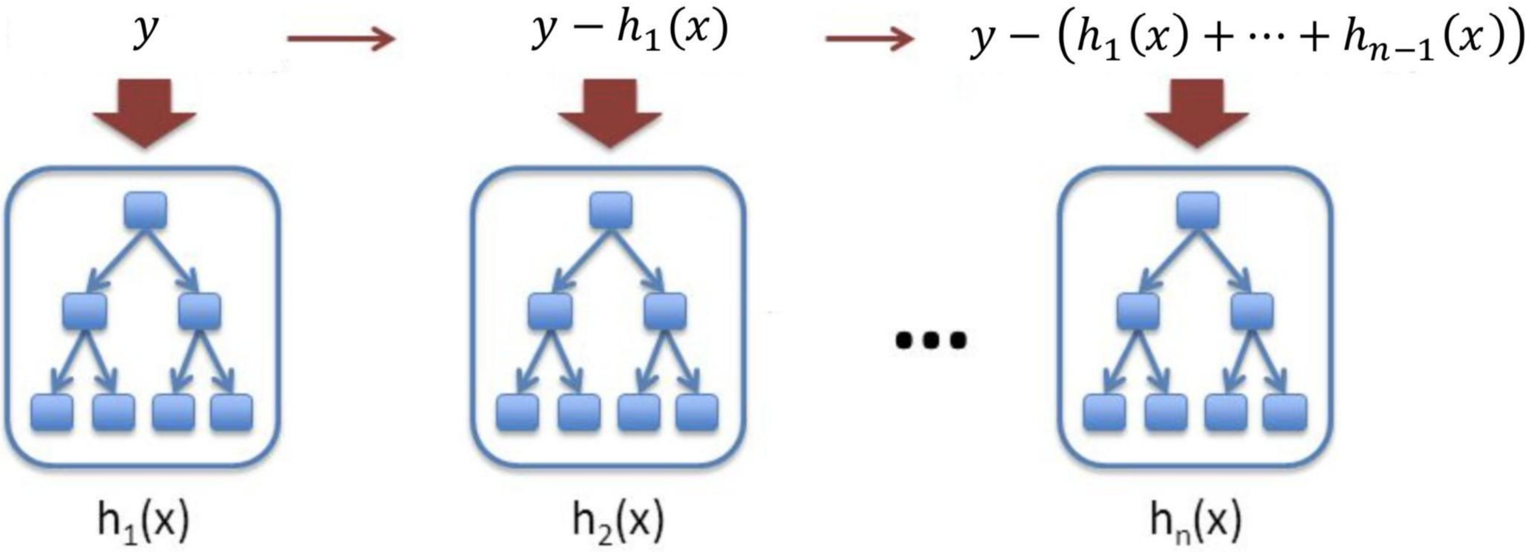
- Correlated models
- Use models with different natures
- Understand your models
- Work with your feature space
- Stack over stacking StackNet or http://ml-ensemble.com/

```
from mlens.ensemble import SuperLearner
ensemble = SuperLearner()
ensemble.add(estimators)
ensemble.add meta(meta estimator)
ensemble.fit(X, y).predict(X)
```

Gradient boosting

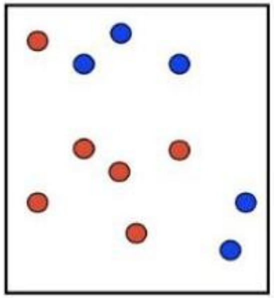
Gradient boosting

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

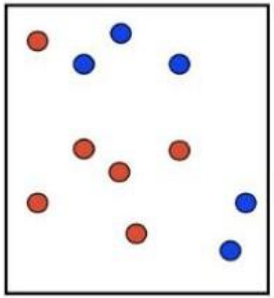


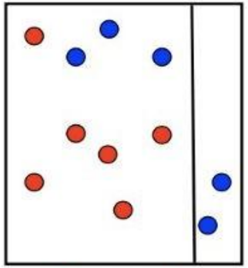
Boosting: intuition

Binary classification problem.
Models - decision stumps.

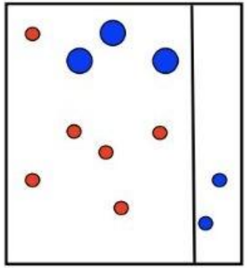


Boosting: intuition

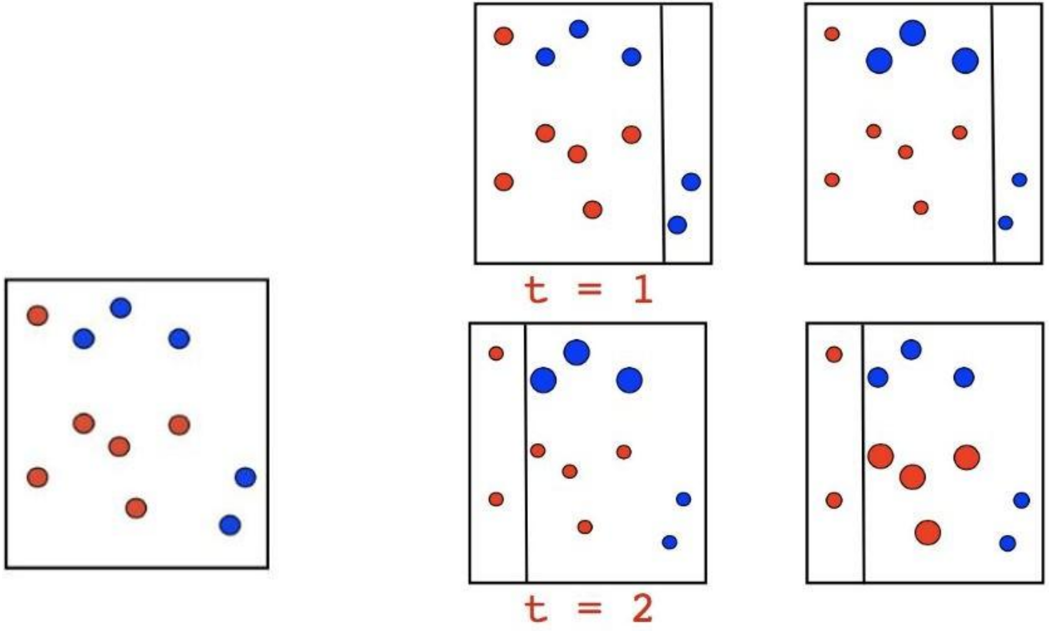




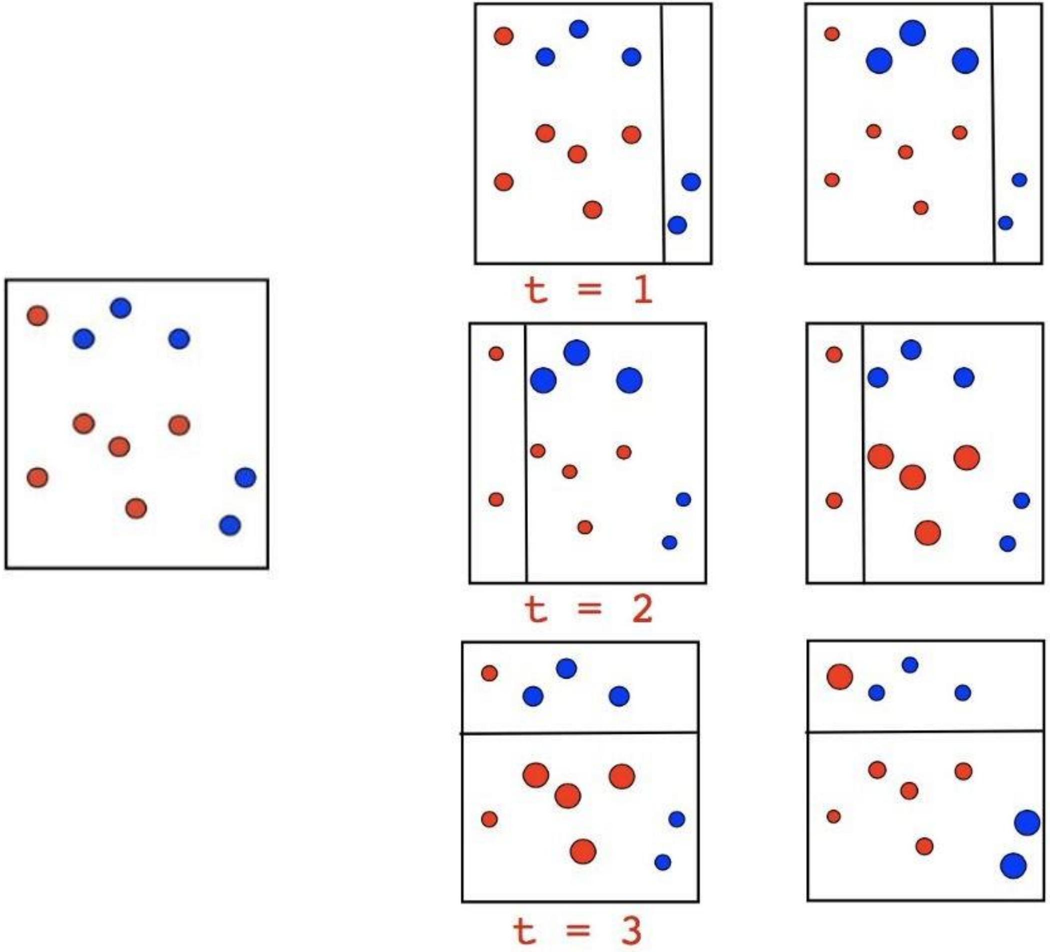
t = 1



Boosting: intuition

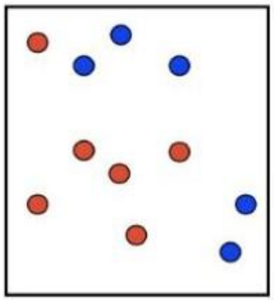


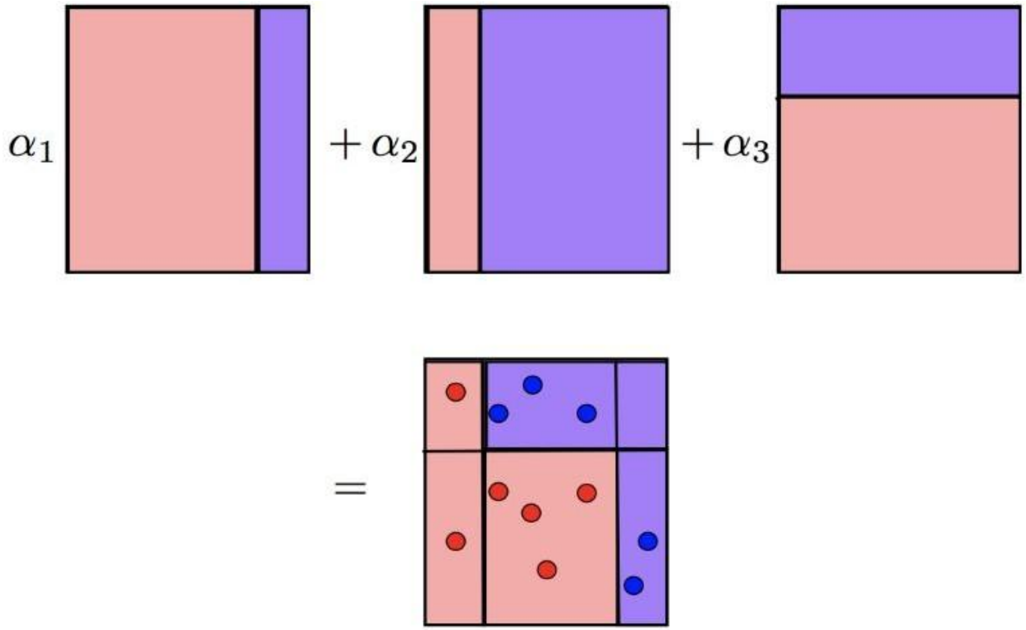
Boosting: intuition



Boosting: intuition

Binary classification problem.
Models - decision stumps.





Boosting: core idea

We train each model in the ensemble so that it corrects the error of the previous one

Boosting: Step by step

Boosting: Step by step regression

$$\frac{1}{2}\sum_{i=1}^{\ell}(a(x_i) - y_i)^2 \to \min_a$$

Result algorithm:

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

$$b_1(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - y_i)^2$$

Boosting: Step by step regression

$$\frac{1}{2}\sum_{i=1}^{\ell}(a(x_i) - y_i)^2 \to \min_a$$

Result algorithm:

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

$$b_1(x) := \arg\min_{b\in\mathcal{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - y_i)^2$$

$$s_i^{(1)} = y_i - b_1(x_i)$$

$$b_2(x) := \arg\min_{b\in\mathcal{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - s_i^{(1)})^2$$

Boosting: Step by step regression

$$\frac{1}{2}\sum_{i=1}^{\ell}(a(x_i)-y_i)^2 \to \min_a$$

Result algorithm:

$$a_N(x)=\sum_{n=1}^{N}b_n(x)$$

$$s_i^{(N)}=y_i-\sum_{n=1}^{N-1}b_n(x_i)=y_i-a_{N-1}(x_i), \qquad i=1,\ldots,\ell;$$

$$b_N(x):=\arg\min_{b\in\mathcal{A}}\frac{1}{2}\sum_{i=1}^{\ell}(b(x_i)-s_i^{(N)})^2$$

Boosting: Step by step regression

$$\frac{1}{2}\sum_{i=1}^{\ell}(a(x_i) - y_i)^2 \to \min_a$$

Result algorithm:

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i), \qquad i = 1, \dots, \ell;$$

$$b_N(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - s_i^{(N)})^2 \quad s_i^{(N)} = y_i - a_{N-1}(x_i) = -\left.\frac{\partial}{\partial z}\frac{1}{2}(z - y_i)^2\right|_{z=a_{N-1}(x_i)}$$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

Find nex algo to minimize the error:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{b_N, \gamma_N}$$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

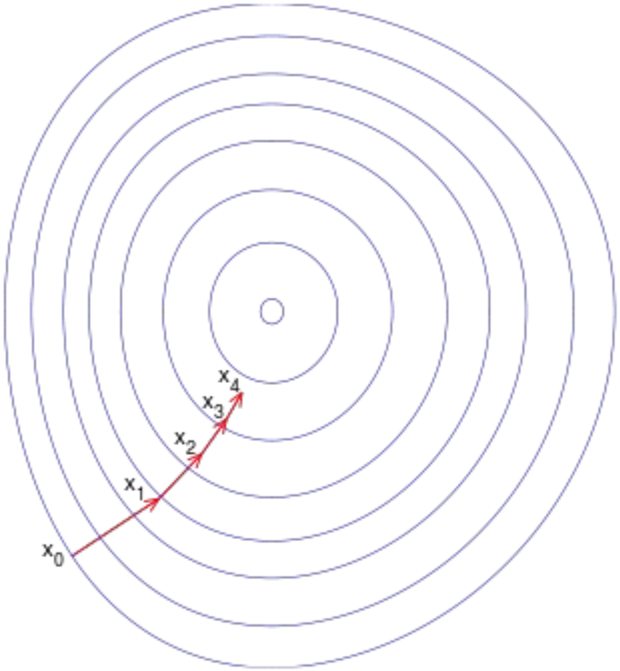
$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

Find nex algo to minimize the error:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{b_N, \gamma_N}$$

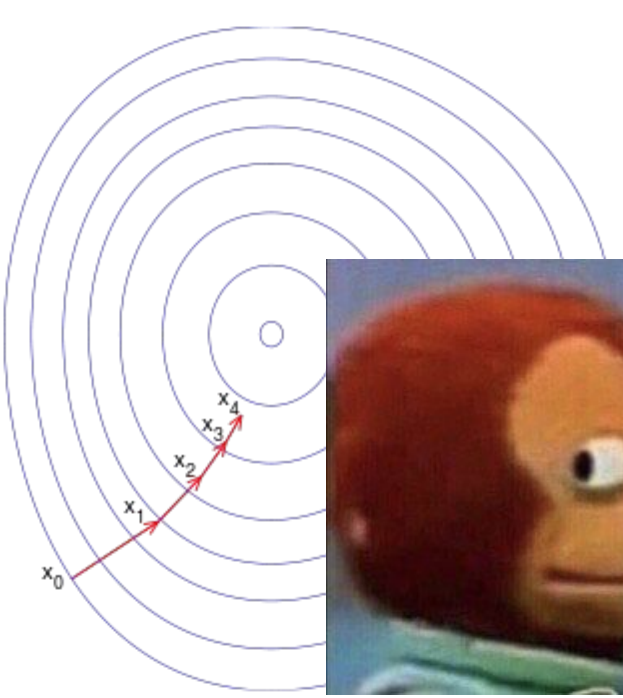
What if we could use gradient descent in *space of our models*?

Gradient boosting: theory



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

Gradient boosting: theory



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

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s_1,\dots,s_\ell}$$

$$b_0(x) = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i$$

$$s_i = -\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}$$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$.

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s_1, \dots, s_\ell}$$

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

$$s_i = -\left.\frac{\partial L}{\partial z}\right|_{z = a_{N-1}(x_i)}$$

For every object:

$$\left(-\left.\frac{\partial L}{\partial z}\right|_{z = a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i)\Big|_{z_i = a_{N-1}(x_i)}$$

Gradient boosting

Denote dataset $\{(x_i, y_i)\}$, loss function $L(y, f)$

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s_1,\dots,s_\ell}$$

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

$$s_i = -\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}$$

For every object: $$\left(-\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}\right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i)\Big|_{z_i=a_{N-1}(x_i)}$$

$$\gamma_N = \arg\min_{\gamma\in\mathbb{R}} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

Gradient boosting: sum up

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: regularization

Add regularizer $$a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x), \qquad \eta \in (0, 1]$$

Number of iteration

Weak learners

Gradient boosting: regularization

Add regularizer $$a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x), \qquad \eta \in (0, 1]$$

Number of iteration

Weak learners

Stochastic Gradient Boosting - train b_i on subsample of data

Gradient boosting: Loss Function

Classification

Logistic Loss

$$L(y, z) = \log(1 + \exp(-yz)).$$

$$b_N = \arg\min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} \left(b(x_i) - \frac{y_i}{1 + \exp(y_i a_{N-1}(x_i))} \right)^2$$

Gradient boosting: Loss Function

Classification

Logistic Loss

$$L(y, z) = \log(1 + \exp(-yz))$$

$$b_N = \arg\min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} \left(b(x_i) - \frac{y_i}{1 + \exp(y_i a_{N-1}(x_i))} \right)^2$$

Regression

MSE

MAE

Where are Trees?

Gradient boosting on Trees

$$b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j]$$

$j = 1, \ldots, J_n$ number of leafs

R_j subspace

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j]$$

Gradient boosting on Trees

$$b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j]$$

$j = 1, \ldots, J_n$ number of leafs

R_j subspace

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j] = a_{N-1}(x) + \sum_{j=1}^{J_N} \gamma_N b_{Nj}[x \in R_j].$$

Gradient boosting on Trees

$$b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j]$$

$j = 1, \ldots, J_n$ number of leafs

R_j subspace

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j] = a_{N-1}(x) + \sum_{j=1}^{J_N} \gamma_N b_{Nj}[x \in R_j].$$

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + \sum_{j=1}^{J_N} \gamma_{Nj}[x \in R_j]\right) \to \min_{\{\gamma_{Nj}\}_{j=1}^{J_N}}$$

Gradient boosting on Trees

$$b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j]$$

$j = 1, \ldots, J_n$ number of leafs

R_j subspace

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j] = a_{N-1}(x) + \sum_{j=1}^{J_N} \gamma_N b_{Nj}[x \in R_j].$$

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + \sum_{j=1}^{J_N} \gamma_{Nj}[x \in R_j]\right) \to \min_{\{\gamma_{Nj}\}_{j=1}^{J_N}}$$

$$\gamma_{Nj} = \arg\min_{\gamma} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma).$$

AdaBoost

$$L(y, z) = e^{-yz}$$

$$L(a, X) = \sum_{i=1}^{\ell} \exp\left(-y_i \sum_{n=1}^{N} \gamma_n b_n(x_i)\right)$$

$$s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=a_{N-1}(x_i)} = y_i \underbrace{\exp\left(-y_i \sum_{n=1}^{N-1} \gamma_n b_n(x_i)\right)}_{w_i}.$$

Extreme Gradient Boosting (XGBoost)

XGBoost

XGBoost

$$s = \left(- \frac{\partial L}{\partial z} \bigg|_{z=a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i) \bigg|_{z_i=a_{N-1}(x_i)}$$

$$b_N(x) = \arg\min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} \left(b(x_i) - s_i\right)^2$$

XGBoost

$$s = \left(- \left. \frac{\partial L}{\partial z} \right|_{z=a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i) \Big|_{z_i=a_{N-1}(x_i)}$$

$$\boxed{b_N(x) = \arg\min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} \left(b(x_i) - s_i\right)^2}$$

Why?

XGBoost

$$s = \left(- \frac{\partial L}{\partial z} \Big|_{z=a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i) \Big|_{z_i=a_{N-1}(x_i)}$$

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + b(x_i)) \to \min_b$$

- the direction calculated by taking into account the second derivatives of the loss function.
- penalties are added for the number of leaves
- criterion of informativeness, dependent on the optimal displacement vector.
- the stop criteria in the training of the tree depends on the shift

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

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 thanks for materials to Radoslav Neychev

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

Links

1. Detailed description of bootstrapping procedure: link
2. Dyakonov blogpost about Gini coefficient and Gini Impurity: link
3. ODS ML course lesson about kNN and Decision Trees: link
4. Habr post about entropy in trees: link
5. Simply about bootstrapping on Habr: link
6. Notes about Decision Trees by Evgeny Sokolov: link