FORNAX

XGBoost - eXtreme Gradient Boosting

Andrzej Pisarek, May 2017

WWW.FORNAX.AI

What is the XGBoost?

https://arxiv.org/pdf/1603.02754.pdf - XGBoost: A Scalable Tree Boosting System (2016)

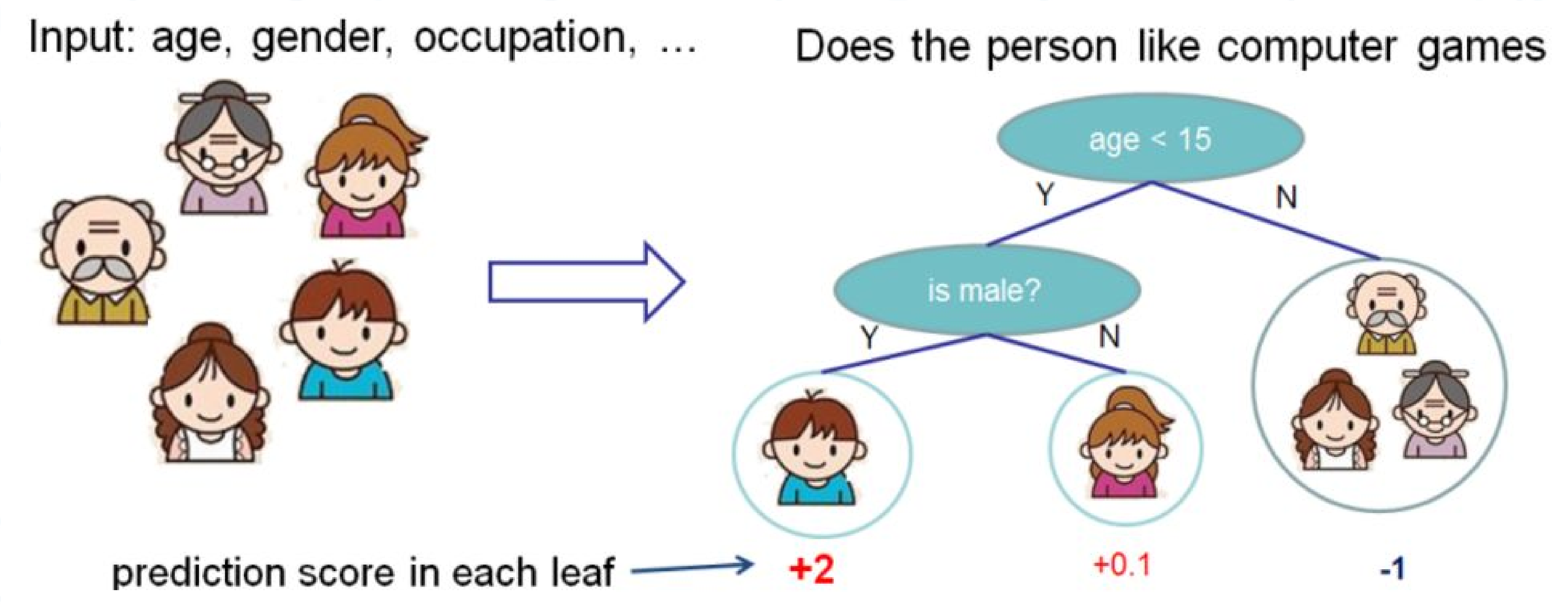
- **17 of 29 winning solutions on Kaggle in 2015 used XGBoost**
- **Contestants analysed:**
  - high energy physics
  - malware
  - ad-click through rate
  - MOOC dropouts
  - customer behaviour
  - store sales

What does the XGBoost consist of?

- Mostly done on trees - *TREE*
- Uses ensemble of weak learners - *BOOSTING*
- Gradient is used to built new weak learner - *GRADIENT*
- It has outstanding implementation - *EXTREME*

Decision tree

https://xgboost.readthedocs.io/en/latest/ - images



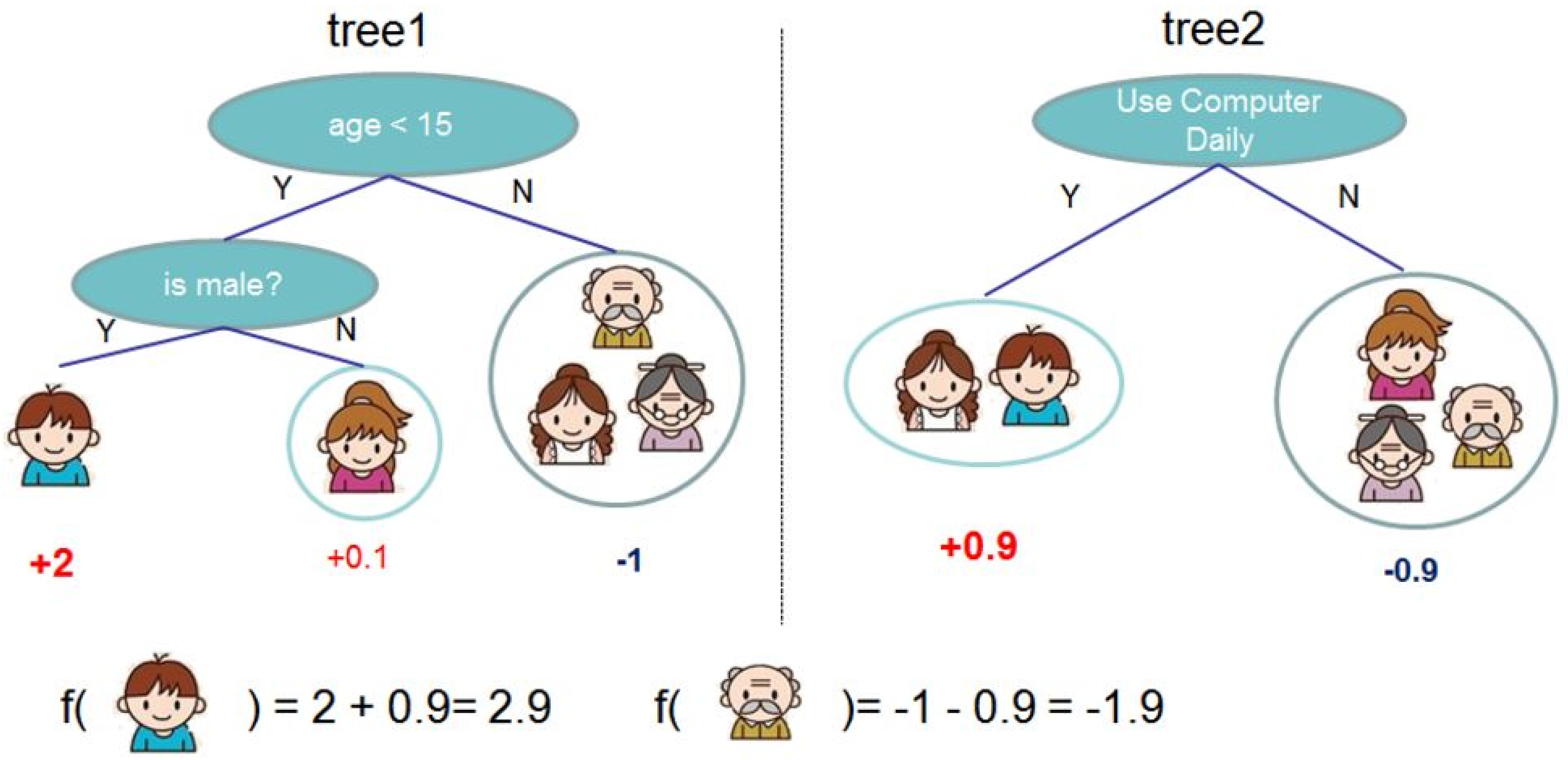
Decision tree

CART style; Classification And Regression Tree

```python
def is_gamer_1(sample):
    # simple tree illustration
    if sample['age'] < 15:
        if sample['gender'] == 'male'
            return 2.0
        else:
            return 0.2

    else:
        return -1
```

Decision tree

CART style; Classification And Regression Tree



Decision tree

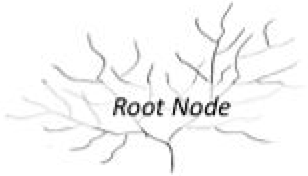
CART style; Classification And Regression Tree

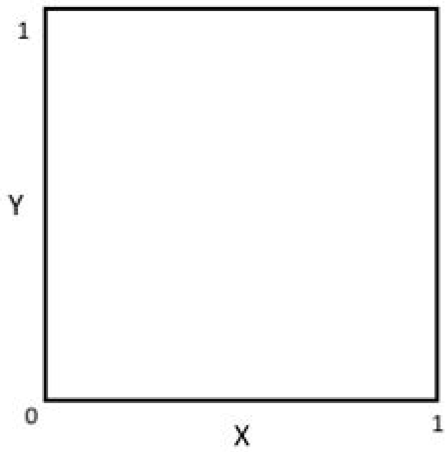
```python
def is_gamer_1(sample):
    # simple tree illustration
    if sample['age'] < 15:
        if sample['gender'] == 'male'
            return 2.0
        else:
            return 0.2

    else:
        return -1
```

```python
def is_gamer_2(sample):
    if sample['use_computer_daily']:
        return 0.9
    else:
        return -0.9

trees = [
    is_gamer_1,
    is_gamer_2
]

def is_gamer_ensemble(sample):
    return sum(tree(sample) for tree in trees) > 0
```

Decision tree

How does the tree grow?






```python
def is_gamer(sample):
    return False
```

```python
def is_gamer(sample):
    if sample['age'] < 15:
        return True
    else:
        return False
```

```python
def is_gamer(sample):
    if sample['age'] < 15:
        return True
    else:
        if sample['gender'] == 'male':
            return True
        else:
            return False
```

For more tutorials: annalyzin.wordpress.com

Decision trees

Why are we using them?

- no need for normalization like in linear models
- good for data of “mixed type”
- outlier-resistant
- scalable for big dataset

Boosting

aka additive learning, a metaalgorithm for doing specific ensembling

- uses weak learner (weak learner is a model that is slightly better than random guess)
- assumes that ensemble of weak learners is a strong learner
- is a sequential algorithm; we add new weak learner to existing ensemble -> we have new ensemble
- each new weak learner should “specialize” in things that previous ensemble gave us errors on
- weak learner usually has high bias and low variance (see bias-variance tradeoff)

Boosting

How does it work?

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

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

$$\text{obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

Gradient boosting

How does it work?

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$
$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

Gradient

Omega stuff

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

$$\sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

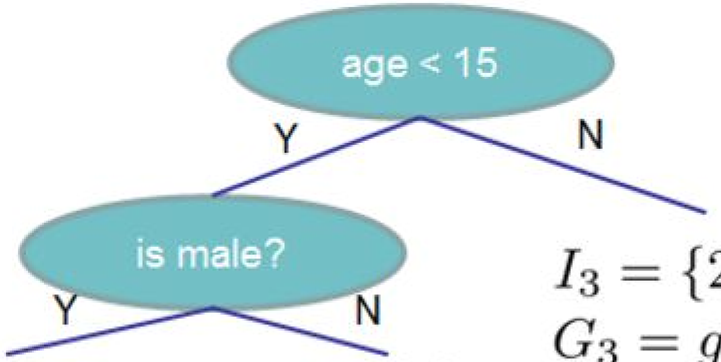
$$\text{obj}^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad \text{obj}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Gradient

Omega stuff

Instance index	gradient statistics
1	g1, h1
2	g2, h2
3	g3, h3
4	g4, h4
5	g5, h5



$I_1 = \{1\}$
$G_1 = g_1$
$H_1 = h_1$

$I_2 = \{4\}$
$G_2 = g_4$
$H_4 = h_4$

$I_3 = \{2, 3, 5\}$
$G_3 = g_2 + g_3 + g_5$
$H_3 = h_2 + h_3 + h_5$

$$Obj = -\sum_j \frac{G_j^2}{H_j+\lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Gradient

In practice, we grow the tree greedily

- Start from tree with depth 0
- For each leaf node of the tree, try to add a split. The change of objective after adding the split is

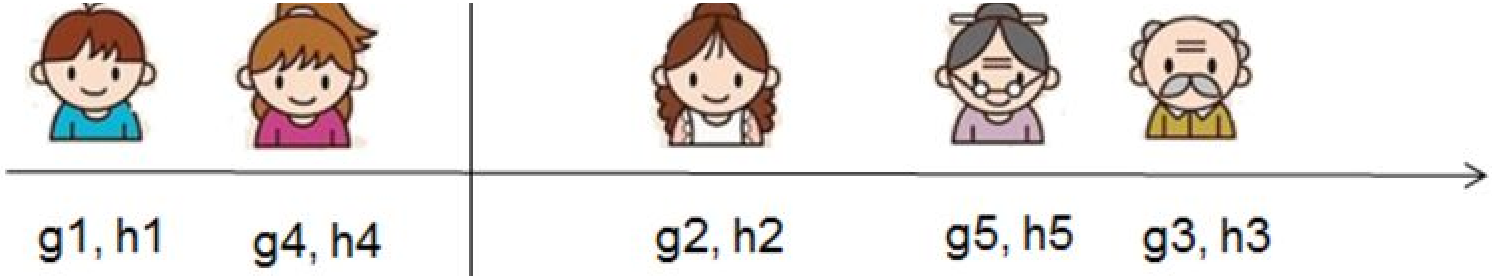
The complexity cost by introducing additional leaf

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

the score of left child

the score of right child

the score of if we do not split



$G_L = g_1 + g_4$

$G_R = g_2 + g_3 + g_5$

Gradient

Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

- Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
- ϵ is called step-size or shrinkage, usually set around 0.1
- This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

Extreme

Why is XGBoost extreme?

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

+ cache aware
+ adding your own loss function
+ choosing base learner

FORNAX

Thank you!

Good luck!

Have fun :)

WWW.FORNAX.AI