

Ensemble Methods

Jie Tang

Tsinghua University

March 24, 2020

Overview

1. Introduction

2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost

3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Outline

1. Introduction

2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost

3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Netflix Prize

- Starting from October 2, 2006, Netflix offers a prize of US $1M for the challenge.
- On September 21, 2009, the prize was given to "BellKor's Pragmatic Chaos".
- On July 25, 2009, the team "The Ensemble", a merger of the teams "Grand Prize Team" and "Opera Solutions and Vandelay United" finally won in the last call.

Training data

user	movie	date	score
1	21	5/7/02	1
1	213	8/2/04	5
2	345	3/6/01	4
2	123	5/1/05	4
2	768	7/15/02	3
3	76	1/22/01	5
4	45	8/3/00	4
5	568	9/10/05	1
5	342	3/5/03	2
5	234	12/28/00	2
6	76	8/11/02	5
6	56	6/15/03	4

Test data

user	movie	date	score
1	62	1/6/05	?
1	96	9/13/04	?
2	7	8/18/05	?
2	3	11/22/05	?
3	47	6/13/02	?
3	15	8/12/01	?
4	41	9/1/00	?
4	28	8/27/05	?
5	93	4/4/05	?
5	74	7/16/03	?
6	69	2/14/04	?
6	83	10/3/03	?

Netflix

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				

The winner was an ensemble of ensembles (including BellKor).

- Ensemble methods are the best performers.
- BellKor's Pragmatic Chaos combined various classifiers including regression models, restricted Boltzmann machines, matrix factorization, k-nearest neighbor, etc.

Ensemble methods

- Multiple "base" models (classifiers, regressors), each covers the complete input space
- Each base model is trained on a slightly different train set
- Combine predictions of all models to produce the output
  - Goal: Improve the accuracy of the "base" model

Methods:

- Bagging
- Boosting
- Stacking

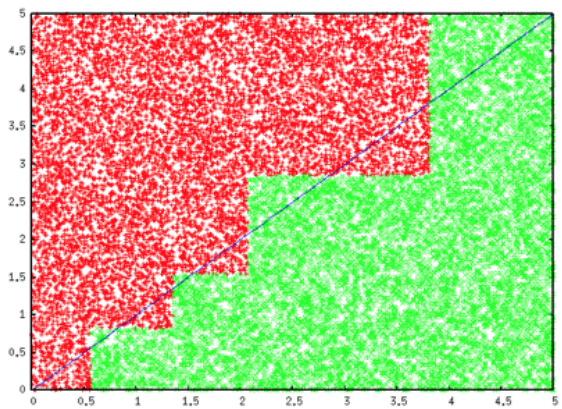
Ensemble methods

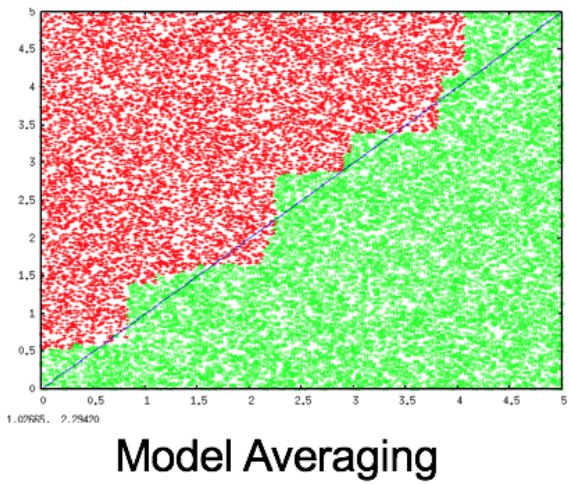
Uncorrelated error reduction

- Suppose we have 5 completely independent classifiers for majority voting
- For a binary classifier, if accuracy is 70% for each
  - $10(.7^3)(.3^2) + 5(.7^4)(.3) + (.7^5)$
  - **83.7% majority vote accuracy**
- 100 such classifiers
  - **99.9% majority vote accuracy**

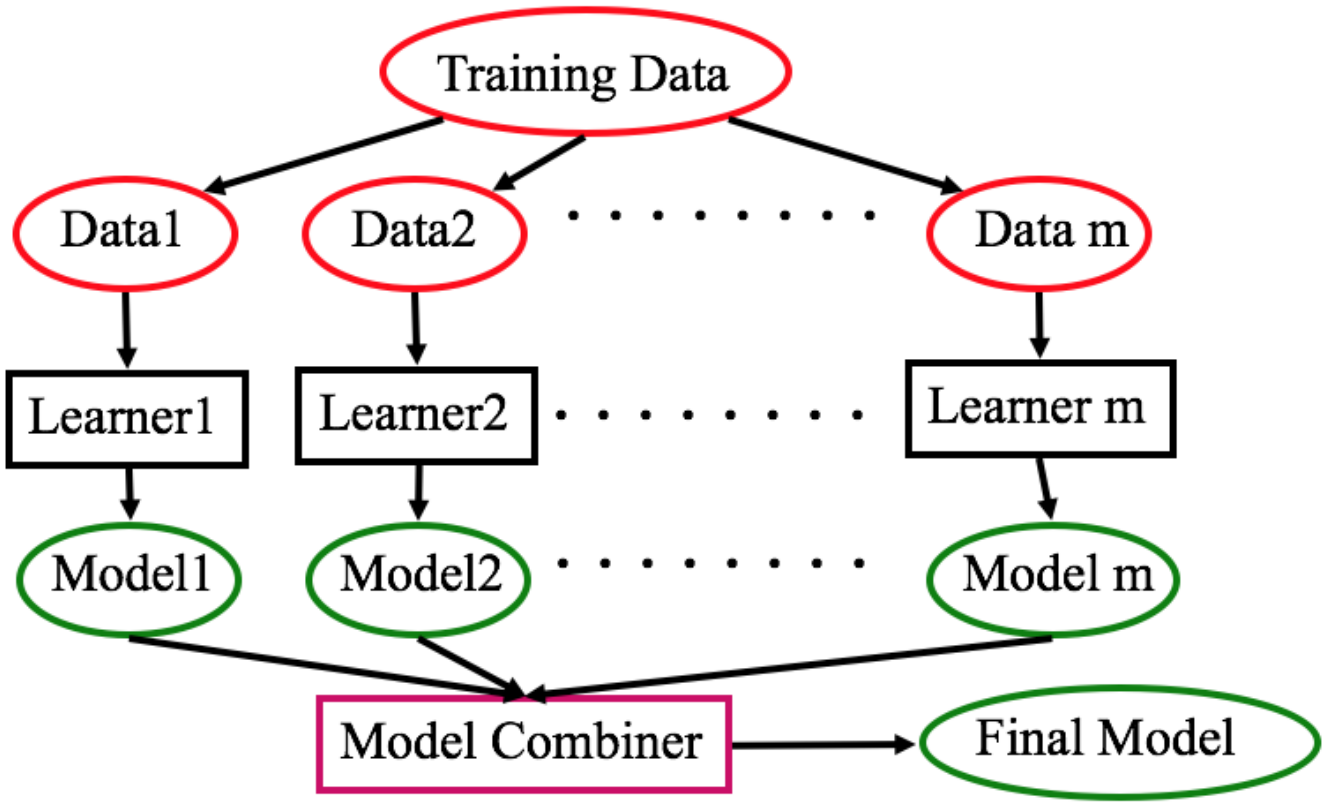
Model Averaging

- Overcome limitations of single hypothesis
  - The target function may not be implementable with individual classifiers, but may be approximated by model averaging





Ensemble methods



Combine decisions of multiple models, e.g. using weighted voting.

Outline

1. Introduction
2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost
3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Ensemble methods

- Classification trees: Often produce noisy or weak classifiers.
- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Schapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forests (Breiman 1999): Fancier version of bagging.

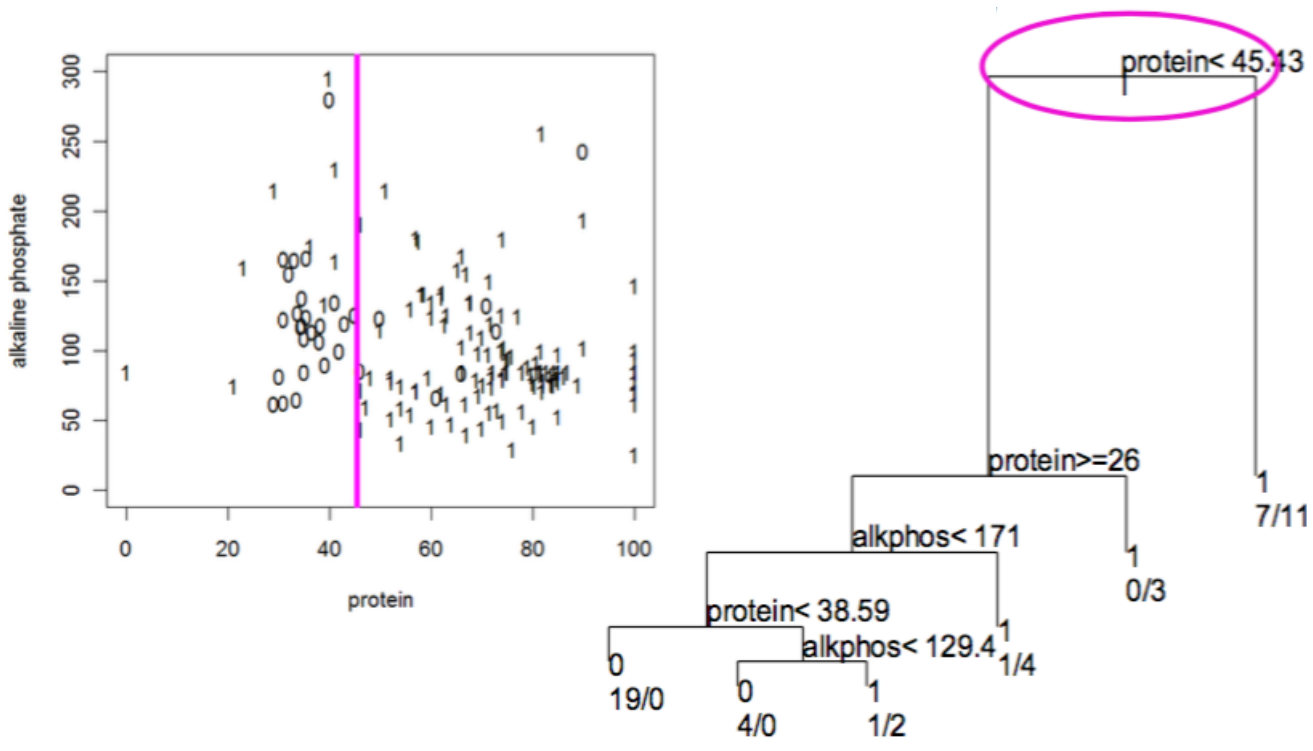
In general, the performance ranking: Boosting $>$ Random Forests $>$ Bagging $>$ Single Tree

Classification Trees

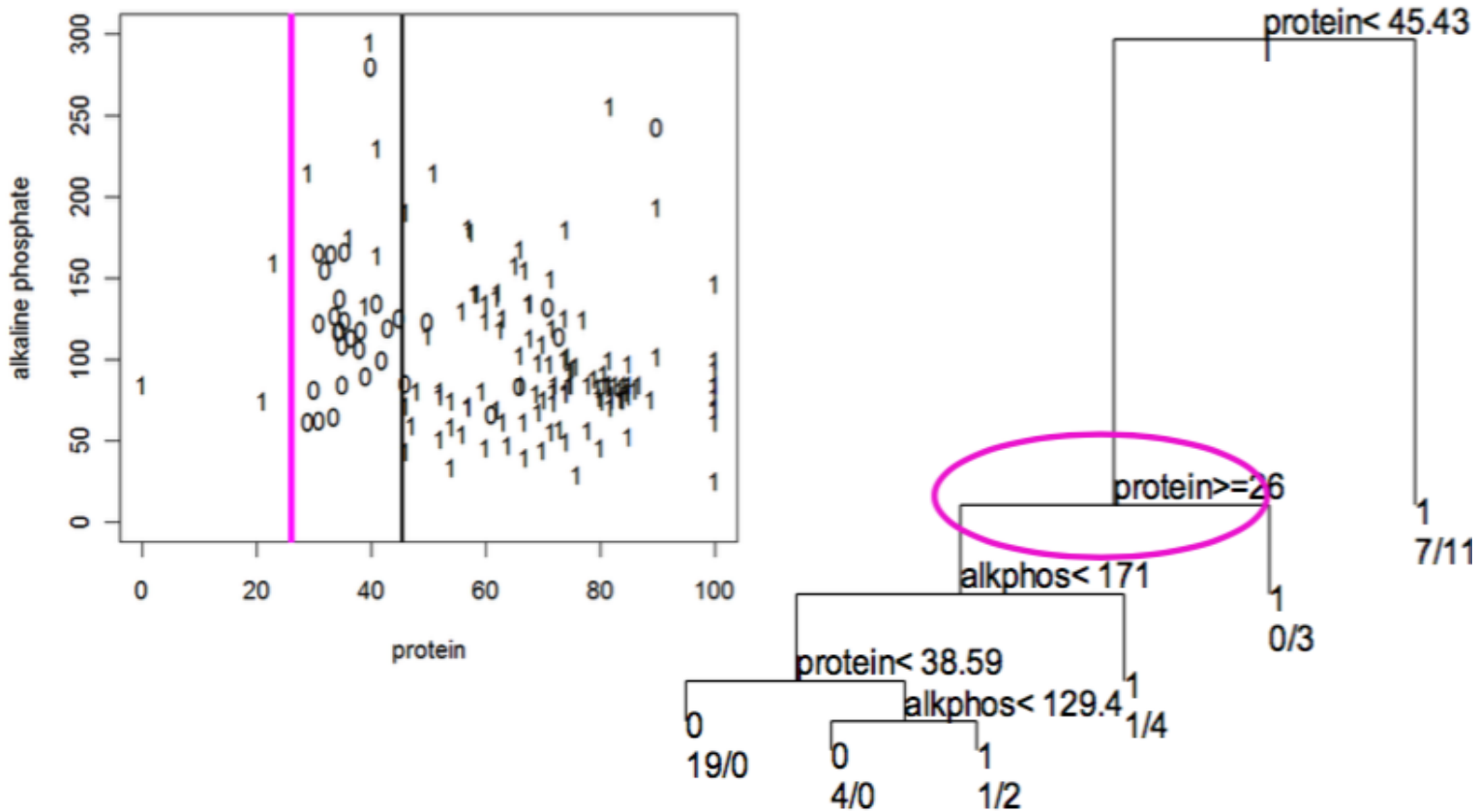
- Represented by a series of binary splits.
- Internal node represent value decision on one of the features, i.e., Is "$feature_1 > 0.5$". Yes goes right, else goes left.
- The terminal nodes are the decision nodes. Typically the class of terminal node is decided by voting.
- The tree is often pruned to an optimal size, evaluated by cross-validation.

Classification Trees Example

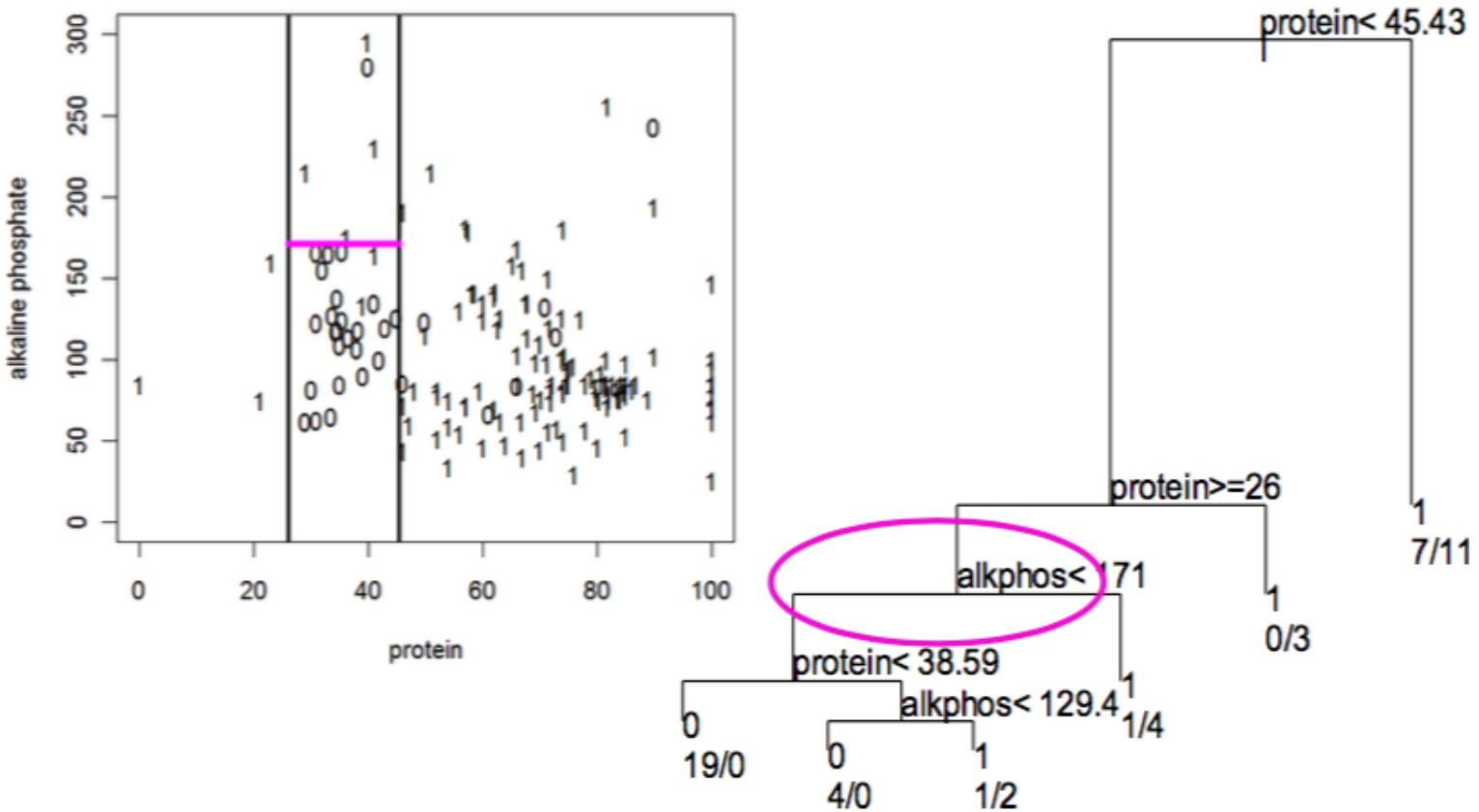
- Predict hepatitis (0=absent, 1=present) using protein and alkaline phosphate.



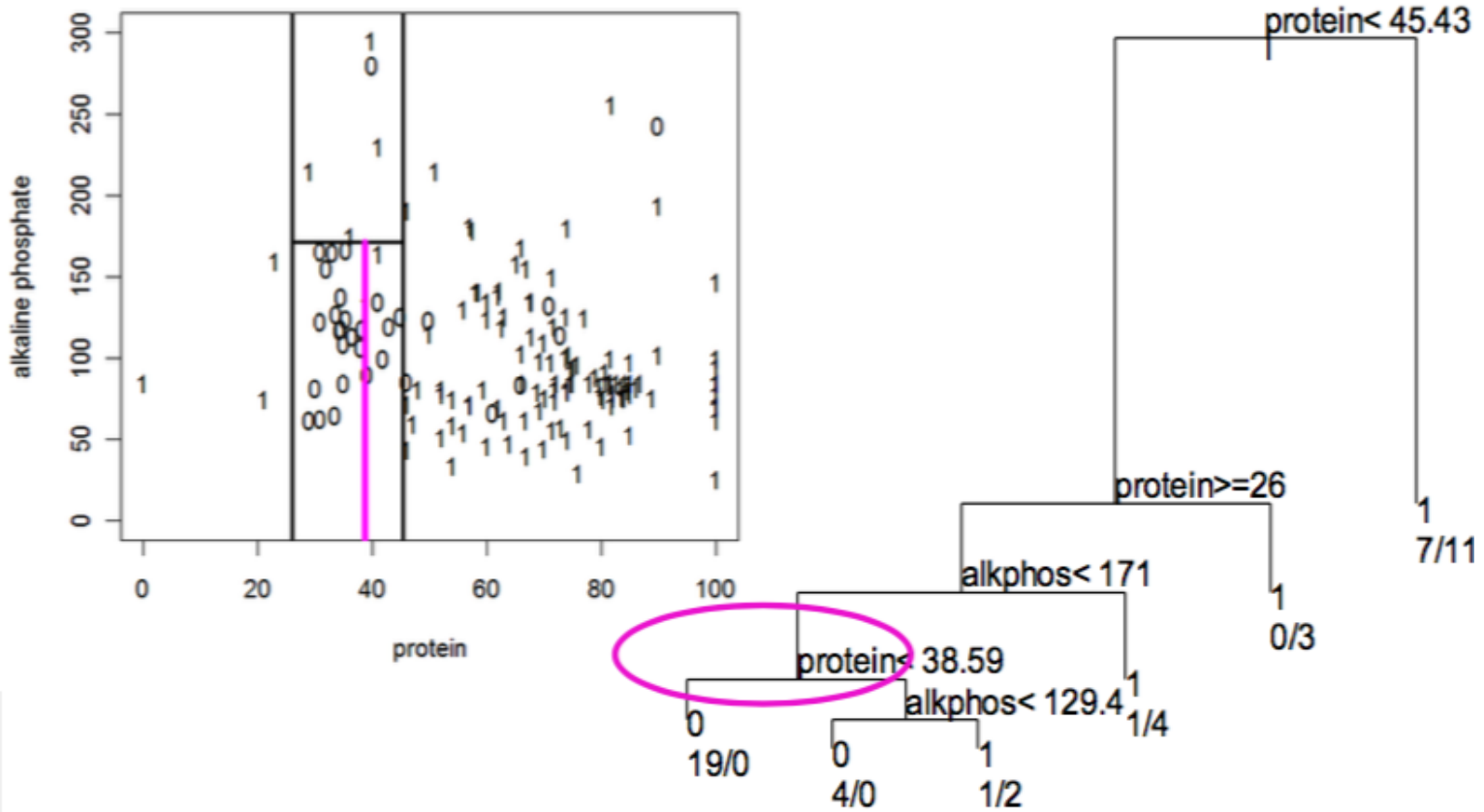
Classification Trees Example



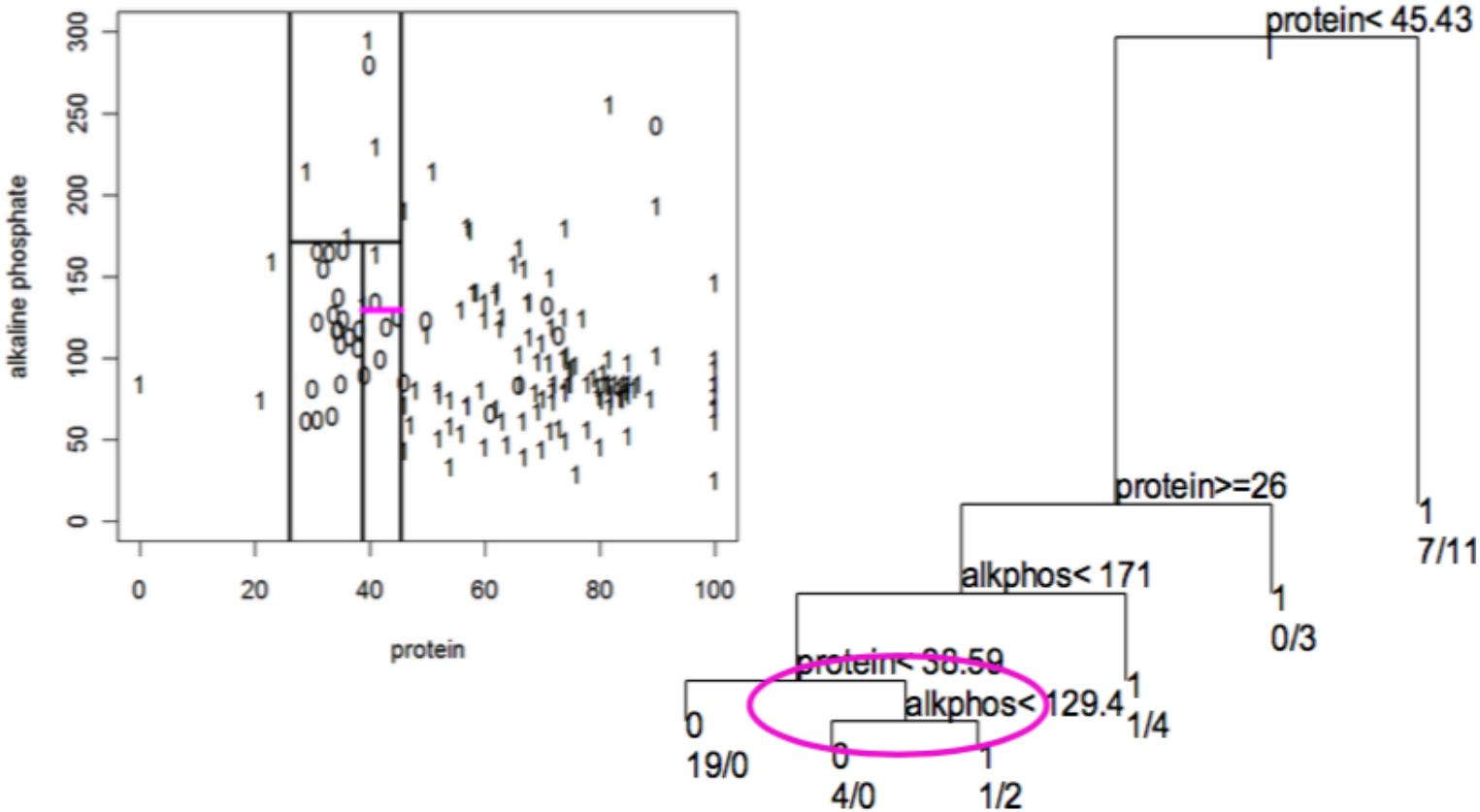
Classification Trees Example



Classification Trees Example



Classification Trees Example



Classification Trees

Advantages

- Handle categorical predictors naturally.
- Computationally simple and quick to fit, even for large problems.
- No formal distributional assumptions (non-parametric).
- Can handle highly non-linear interactions and classification boundaries.
- Very easy to interpret if the tree is small.

Classification Trees

Disadvantages

- Accuracy — methods such as support vector machines and ensemble classifiers often have 30% lower error rates than CART.
- Instability —- if we change the data a little, the tree picture can change a lot. So the interpretation is not as straightforward as it appears.

Today, we can do better!

Ensembling Learning

Outline

1. Introduction

2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost

3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Bagging (**B**ootstrap **Agg**regat**ing**)

Create ensembles by "bootstrap sampling", i.e., repeatedly randomly resampling the training data (Brieman, 1996).
Given that we have N training examples,

- Draw bootstrap samples with replacement to get data subsets, $D^1, \ldots, D^T$ each of size N.
- Train base learners for each dataset.
- Aggregate the outputs by **(weighted) majority voting** (for classification) or **(weighted) averaging** (for regression).

Bagging can reduce the variance of unstable learners (like trees, neural networks), leading to better performance.

Majority Voting

- The final output class label is the one that receives more than half of the votes;
- If none of the class labels receives more than half of the votes, a rejection option will be given

$$H(x) = \begin{cases} y_j & \text{if} \sum_{i=1}^{T} I(C_i(x) = j) > \frac{1}{2} T \\ \text{rejection} & \text{otherwise} \end{cases} \tag{1}$$

Given that classifiers are independent and each has an accuracy p, then the accuracy of the ensemble is

$$P = \sum_{k=\lfloor T/2+1 \rfloor} \binom{T}{k} p^k (1-p)^{T-k} \tag{2}$$

Weighted Voting

- Different classifiers are given different shares of votes. The final output class label is the one that receives more than half of the votes;
- If none of the class labels receives more than half of the votes, a rejection option will be given

$$H(x) = \begin{cases} y_j & \text{if} \sum_{i=1}^{T} I(C_i(x) = j) V(i) > \frac{1}{2} \\ \text{rejection} & \text{otherwise} \end{cases} \quad (3)$$

Given 5 independent binary classifiers with accuracy $(0.7, 0.7, 0.7, 0.9, 0.9)$. Majority voting gives accuracy

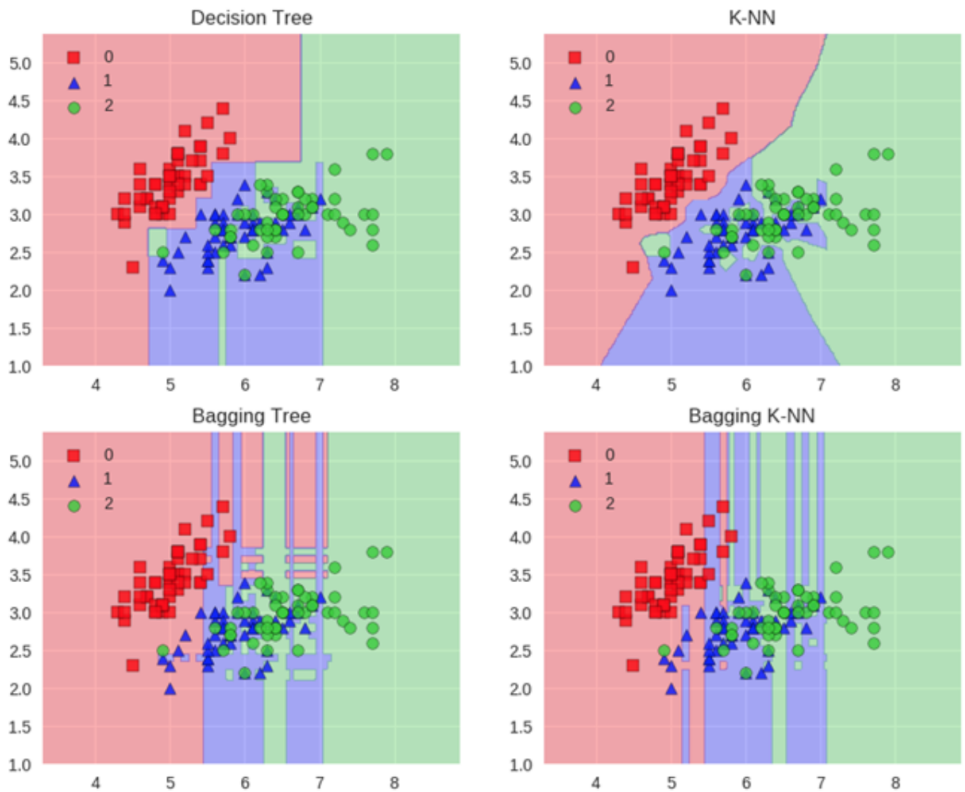
$$P = 0.7^3 + 2*3*0.7^2*0.3*0.9*0.1 + 3*0.7*0.3*0.9^2 = 0.933 \quad (4)$$

Weighted voting with weight $(0.1, 0.1, 0.1, 0.35, 0.35)$ gives

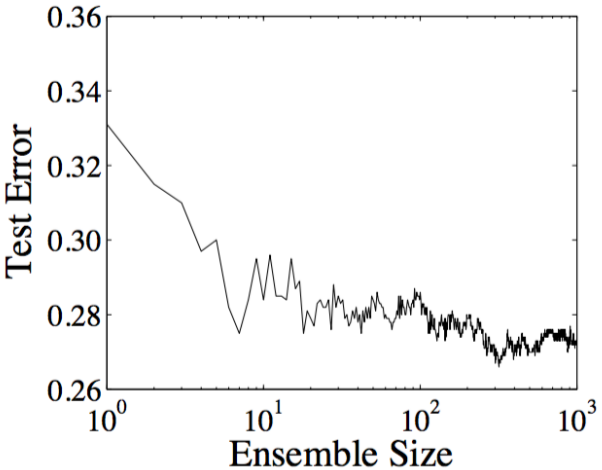
$$P = 0.9^2 + 2*3*0.9*0.1*0.7^2*0.3 + 2*0.9*0.1*0.7^3 = 0.951 \quad (5)$$

Bagging

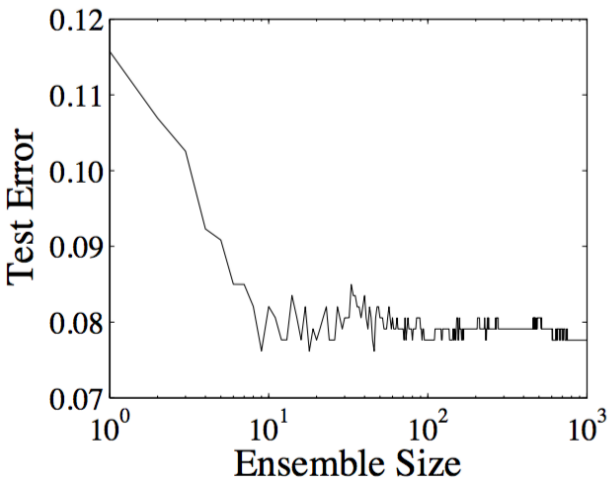
- The decision tree bagging ensemble achieved higher accuracy in comparison to the K-NN bagging ensemble. K-NNs are called stable learners.



Ensemble Size



(a) credit-g



(b) soybean

- Bagging on two UCI data sets (credit-g, soybean).
- The performance of Bagging converges as the ensemble size, i.e., the number of base learners, grows large.

*UCI Machine Learning Repository: Data Sets
https://archive.ics.uci.edu/ml/datasets.html

Bias-Variance Decomposition

- The generalization error of a learner can be divided into three components, i.e., intrinsic noise, bias and variance (intrinsic noise is often subsumed into the bias term.)
- Expected error = Bias + Variance

$$\begin{aligned} \mathbb{E}\left[(h-f)^2\right] &= (\mathbb{E}[h]-f)^2 + \mathbb{E}\left[(h-\mathbb{E}[h])^2\right] \\ &= bias(h)^2 + variance(h) \end{aligned} \tag{6}$$

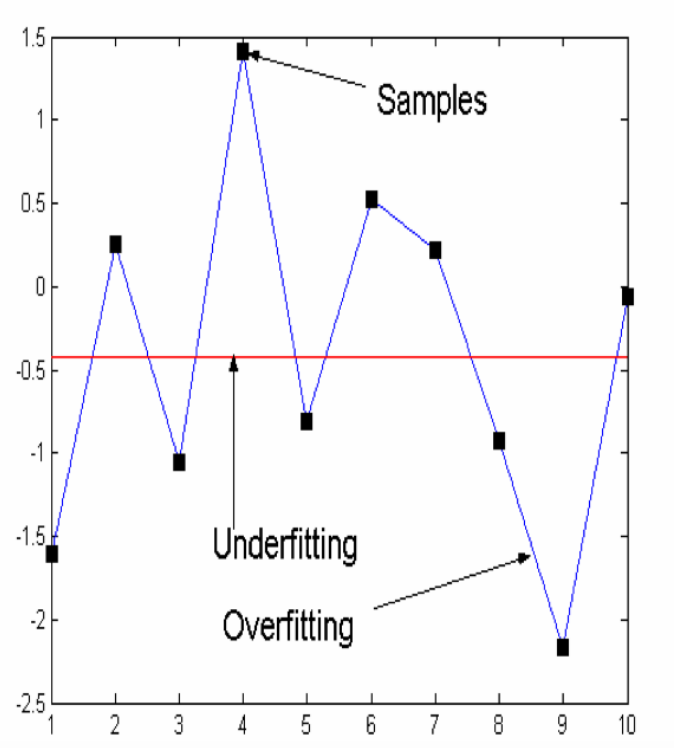
h is the estimated function and f is true function

Question

Q: Try to explain why Bagging is difficult to improve the performance of Naive Bayes classifier.

When Bagging works?

For a simple (a.k.a. weak) learners e.g., naive Bayes, logistic regression, decision stumps (or shallow decision trees),



- Under-fitting:
  - High bias
  - Small variance (smaller influence of examples in the training set)
- Over-fitting:
  - Small bias (models flexible enough to fit well to training data)
  - Large variance (models depend very much on the training set)

Averaging decreases variance

- Example
  - Assume we have a random variable x with a $N(\mu, \sigma^2)$ distribution.
  - If variable x is measured K times $(x_1, x_2, \ldots, x_K)$, and the esitmator is $(x_1 + \cdots + x_K)/K$,
    - Mean of the estimate is still μ
    - But, variance is smaller:

$$variance[(x1 + \Delta\Delta\Delta + x_K)/K] = \sigma^2/K$$

- Bagging is a kind of averaging!
- Bagging decreases variance of the base model without changing the bias.
- Bagging decreases variance to σ^2/K only when base learners are independent, but base learners are usually correlated.

Covariance

Suppose we have k weak learners, each weak learner h_i makes an error err_i on each example. The errors are drawn from a multivariate normal distribution with $\mathbb{E}(err_i^2) = v$ and $\mathbb{E}(err_i \cdot err_j) = c$, The expected squared error of the average predictor is

$$\begin{aligned} \mathbb{E}\left[(\frac{\sum_i err_i}{k})^2\right] &= \frac{1}{k^2}\mathbb{E}\left[\sum_i err_i^2 + \sum_i \sum_{j \neq i} err_i \cdot err_j\right] \\ &= \frac{1}{k}v + \frac{k-1}{k}c \end{aligned} \tag{7}$$

- The smaller the correlation, the better the ensemble.
  - if the errors are perfectly correlated ($c = v$), the squared error is still v.
  - If the errors are perfectly uncorrelated ($c = 0$), the expected squared error of the ensemble is only $\frac{v}{k}$

Outline

1. Introduction
2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost
3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Random Forests

RF is refinement of bagged trees; quite popular.

Given that we have p features,

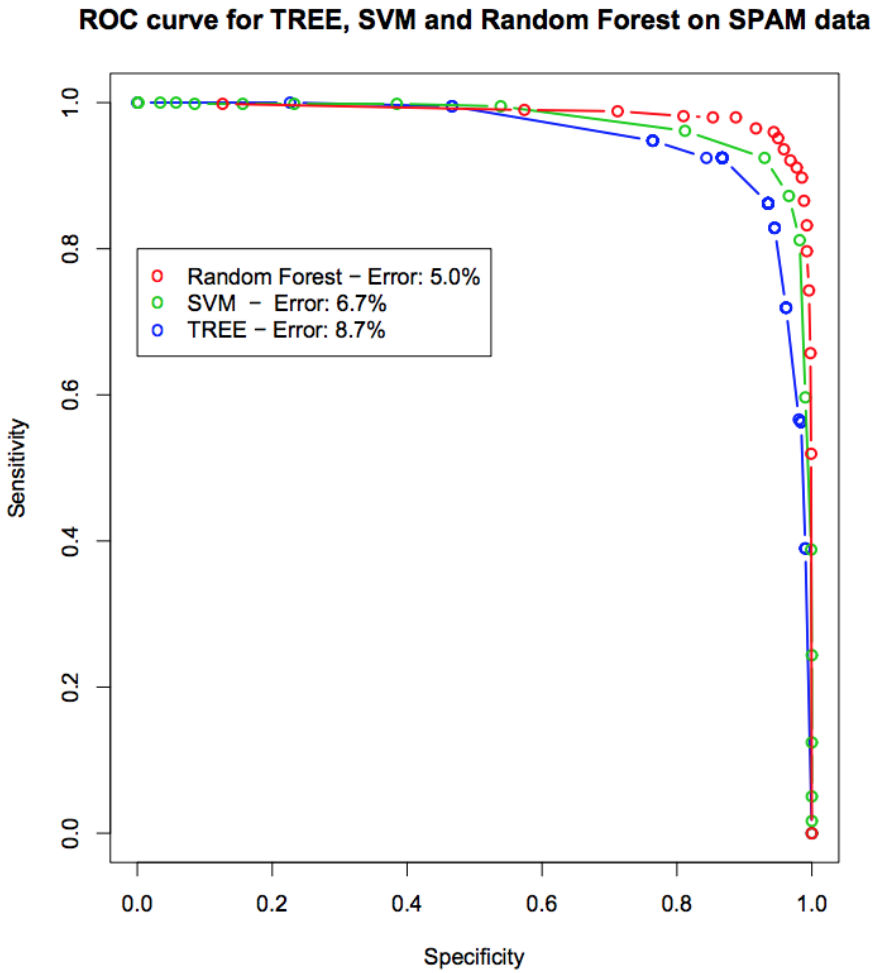
- for each tree,
  - choose a sub dataset by choosing N times (N is the number of training examples) with replacement from the training set.
  - at each tree split, randomly select m features, and split only on these m features. $m = \sqrt{p}$ or $\log_2 p$ typically.
- use majority voting among all the trees
- compared to bagging, RF improves by "de-correlating" tree learners.

Random Forests

- Grow each tree as deep as possible, no pruning.
- Out-of-bag data can be used to estimate cross-validation error machine learning methods.
  - For each training point, get prediction from averaging trees where point is not included in bootstrap sample.
- Accuracy — Random Forests is competitive with the best known machine learning methods.
- Instability — if we change the data a little, the individual trees may change but the forest is relatively stable because it is a combination of many trees.

Random Forests

Example: Predicting e-mail spam



Random Forest dominates both other methods on the SPAM data — 5.0% error. Used 500 trees with default settings for *random Forest package* in *R*.

Outline

1. Introduction
2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost
3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Boosting

Weak Learner: only needs to generate a hypothesis with a training accuracy greater than 0.5, i.e., $< 50\%$ error over any distribution

- Strong learners are sometimes very difficult to construct;
- Constructing weaker Learners is relatively easy

Questions: Can a set of weak learners create a single strong learner?

YES!

Boost weak classifiers to a strong learner.

Boosting

Instead of sampling (as in bagging) re-weight examples!

- Examples are given weights.
- At each iteration, a new hypothesis is learned (weak learner) and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.
- Final classification based on weighted vote of weak classifiers.

Construct Weak Classifiers

- Using Different Data Distribution
  - Start with uniform weighting
  - During each step of learning
    - Increase weights of the examples which are not correctly learned by the weak learner
    - Decrease weights of the examples which are correctly learned by the weak learner Idea
- Idea: focus on difficult examples which are not correctly classified in the previous steps

Combine Weak Classifiers

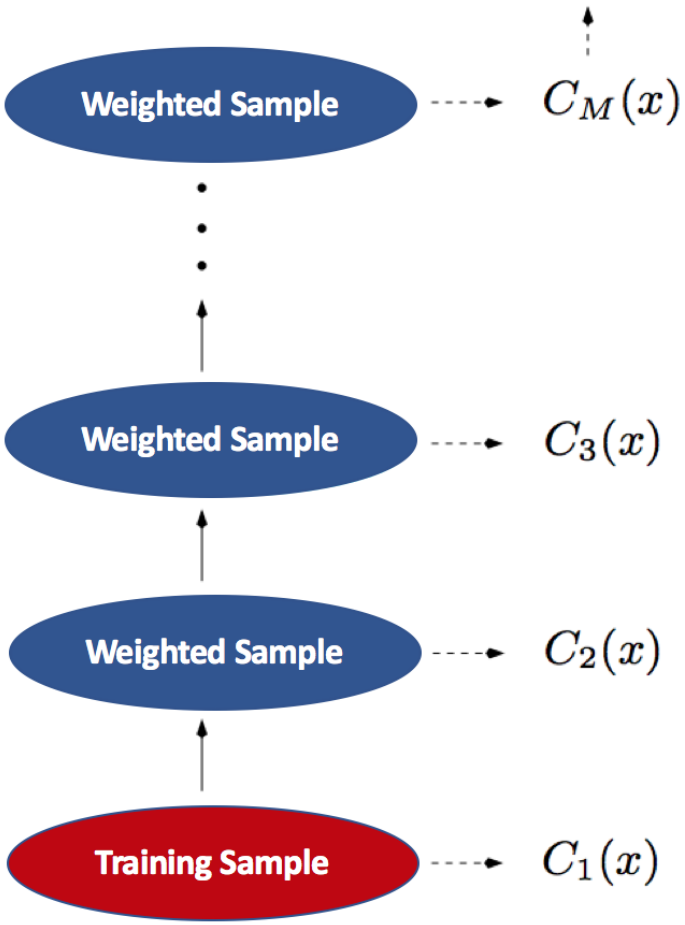
Weighted Voting

- Better weak classifier gets a larger weight
- Iteratively add weak classifiers

Many variants depending on how to set the weights and how to combine the hypotheses.

ADABOOST is quite popular!

Boosting



- Average many trees, each grown to re-weighted versions of the training data.
- Final Classifier is weighted average of classifiers:

$$C(x) = sign[\sum_{m=1}^{M} \alpha_m C_m(x)]$$

AdaBoost (Adaptive Boosting)

1. Initialize the sample weights $\boldsymbol{w} = (\frac{1}{N}, \ldots, \frac{1}{N})$
2. Repeat for $m = 1$ to M,
   - Fit a classifier C_m on training samples weighted by w_i
   - compute the error err_m and weight α_m of classifier C_m

$$\begin{aligned} \text{err}_m &= \frac{\sum_i w_i I(y_i \neq C_m(x_i))}{\sum_i w_i} \\ \alpha_m &= \frac{1}{2} ln \frac{1 - \text{err}_m}{\text{err}_m} \end{aligned} \tag{8}$$

[Freund & Schapire'95]

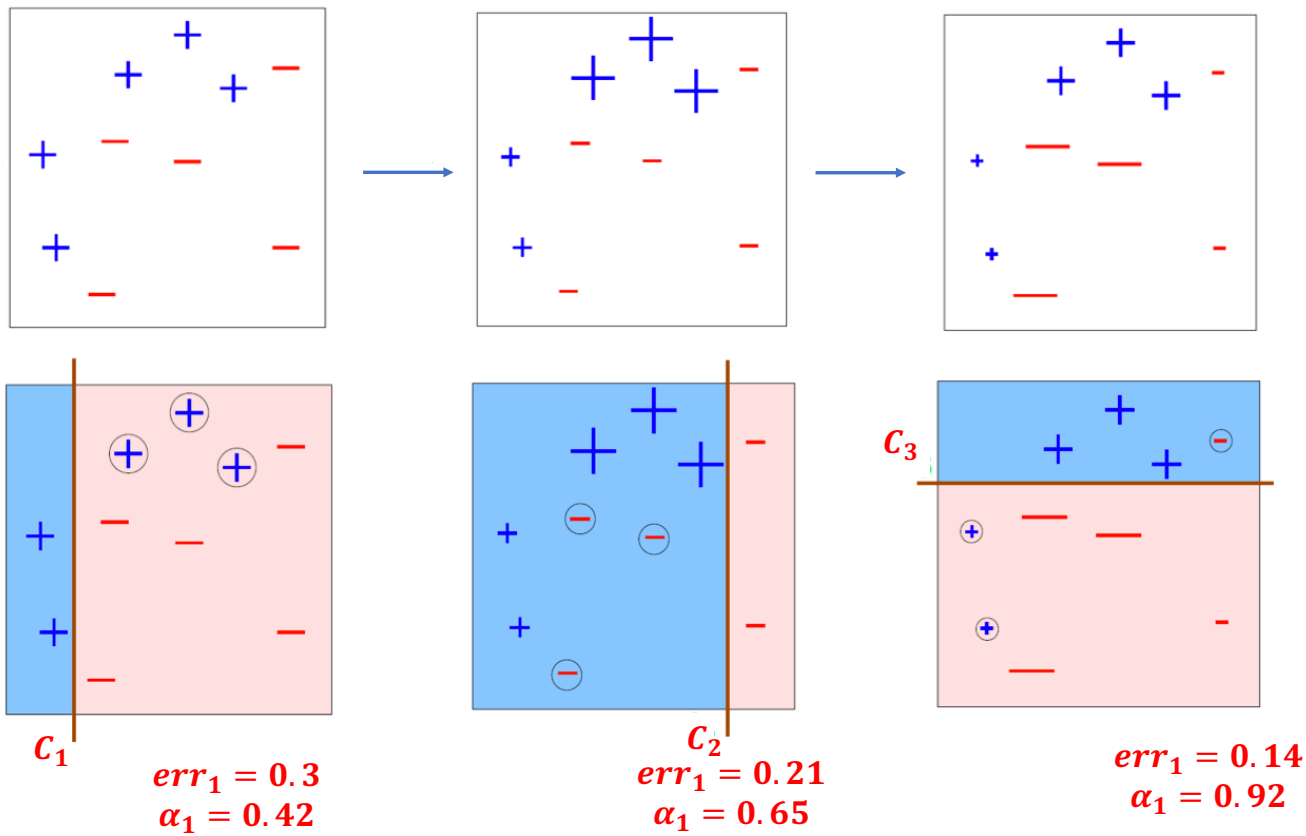
   - Increase weights for mis-classified samples, for $i = 1, \ldots, N$,

$$w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq C_m(x_i))]/Z_i$$

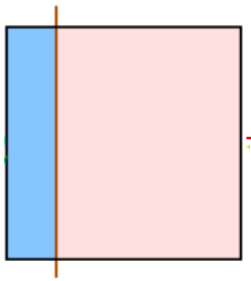
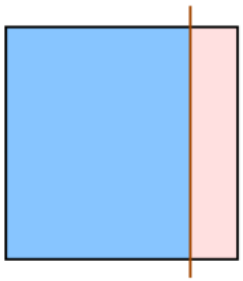
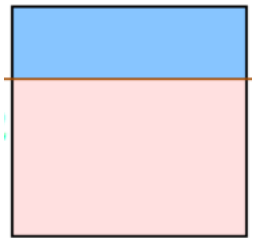
   Z_i renormalize the sum of w_i to 1.

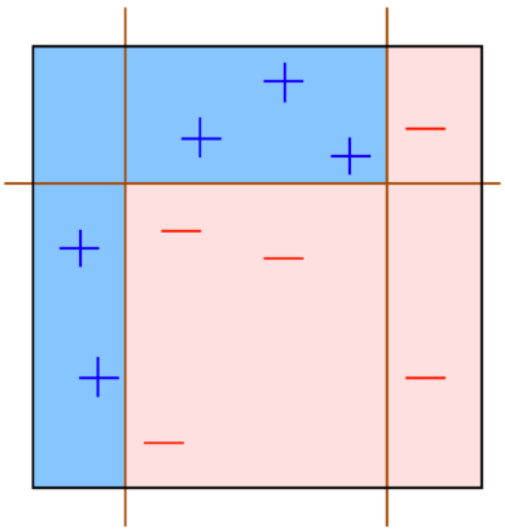
3. Output $C(x) = sign[\sum_{m=1}^{M} \alpha_m C_m(x)]$.

Example (Decision Stumps)



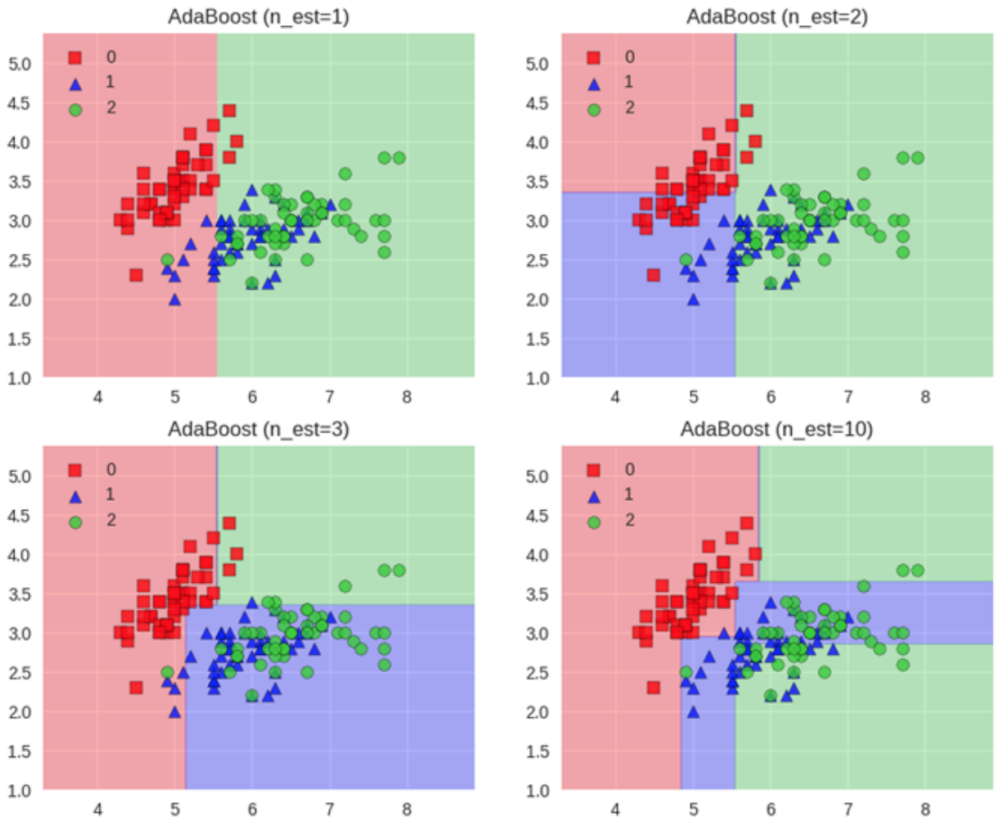
Example (Decision Stumps)

$C_{final} = 0.42$  $+0.65$  $+0.92$ 

$=$ 

Example

- Each base learner consists of a decision tree with depth 1 (ensemble size = 1, 2, 3, 10).



Analyzing Training Error

Analysis in [Freund & Schapire'95] reveals:

- What α_m to choose for classifier m? $\Rightarrow \alpha_m = \frac{1}{2} ln \frac{1-\text{err}_m}{\text{err}_m}$
  - α_m is chosen by minimizing the upper bound of error on each iteration.
- If each weak learner C_m is slightly better than random guessing ($\text{err}_m < 0.5$), then training error of AdaBoost decays exponentially fast in number of rounds M.

$$\underbrace{\frac{1}{N}\sum_{i=1}^{N} I(y_i \neq C(x_i))}_{\text{Training Error}} \leq \exp(-2\sum_{m=1}^{M}(1/2 - \text{err}_m)^2) \qquad (9)$$

Margin Based Bounds

- Another analysis tool [Schapire, Freund, Bartlett, Lee'98]
- $\text{margin}(x_i, y_i) = \frac{y_i \sum_m \alpha_m C_m(x_i)}{\sum_m \alpha_m}$
  - is positive iff $C = sign[\sum_m \alpha_m C_m]$ correctly classifies the example
  - measures confidence in the prediction
- The error bound of ensemble is

$$\text{error}_{\text{true}} \leq Pr\left[\text{margin}(x, y) \leq \theta\right] + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2}}\right) \qquad (10)$$

for any $\theta > 0$ with high probability

  - d — VC dimension of weak learner, measures complexity of classifier.

Margin Based Bounds

$$\text{error}_{\text{true}} \leq Pr\left[\text{margin}(x, y) \leq \theta\right] + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2}}\right) \qquad (11)$$

- Boosting increases the margin aggressively since it concentrates on the examples with the smallest margin.
- Even after the training error reaches zero, boosting continues to increase the margins of the training examples effecting a drop in the test error.

Outline

1. Introduction
2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost
3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Stacking: Learning to Combine

Stacking is also called meta learning

It trains first-level learners, then combines them into a second-level learner by learning.

- Train first level learners—$C_1, \ldots, C_T$ using different algorithms on training dataset D.
- Generate a new dataset D', each data point is represented as $((C_1(x_i), \ldots, C_T(x_i)), y_i)$ for $i = 0, \ldots, N$
- Train a second-level learner C' on D'
- Final output for x_i is $C'(C_1(x_i), \ldots, C_T(x_i))$

Outline

1. Introduction
2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost
3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

XGBoost[1]

XGBoost stands for e**X**treme **G**radient **Boost**ing [Chen T, Guestrin C. Xgboost].

- An open-sourced tool
  - Computation in C++
  - R/python/Julia interface provided
- A variant of the gradient boosting machine
  - Tree-based model
- The winning model for many kaggle competitions

[1] This part refers to Tianqi Chen's slides.

XGBoost

XGBoost stands for e**X**treme **G**radient **Boost**ing.

- Suppose we have K trees, the model makes prediction by

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

  where each f_k is the prediction from a decision tree.
- The prediction at the t-th step can be defined as

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i)$$

How do we learn?

- Objective: $\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- We can not use methods such as SGD, to find f (since they are trees, instead of just numerical vectors)
- Solution: **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

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

Additive Training

- How do we decide which f to add?
  - Optimize the objective!
- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k} \Omega(f_k) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant} \end{aligned}$$

- Consider square loss

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \Omega(f_t) + \text{const} \\ &= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + \text{const} \end{aligned}$$

Taylor Expansion Approximation of Loss

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$ seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f^{'}(x)\Delta x + \frac{1}{2} f^{''}(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$

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

- Think of square loss

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

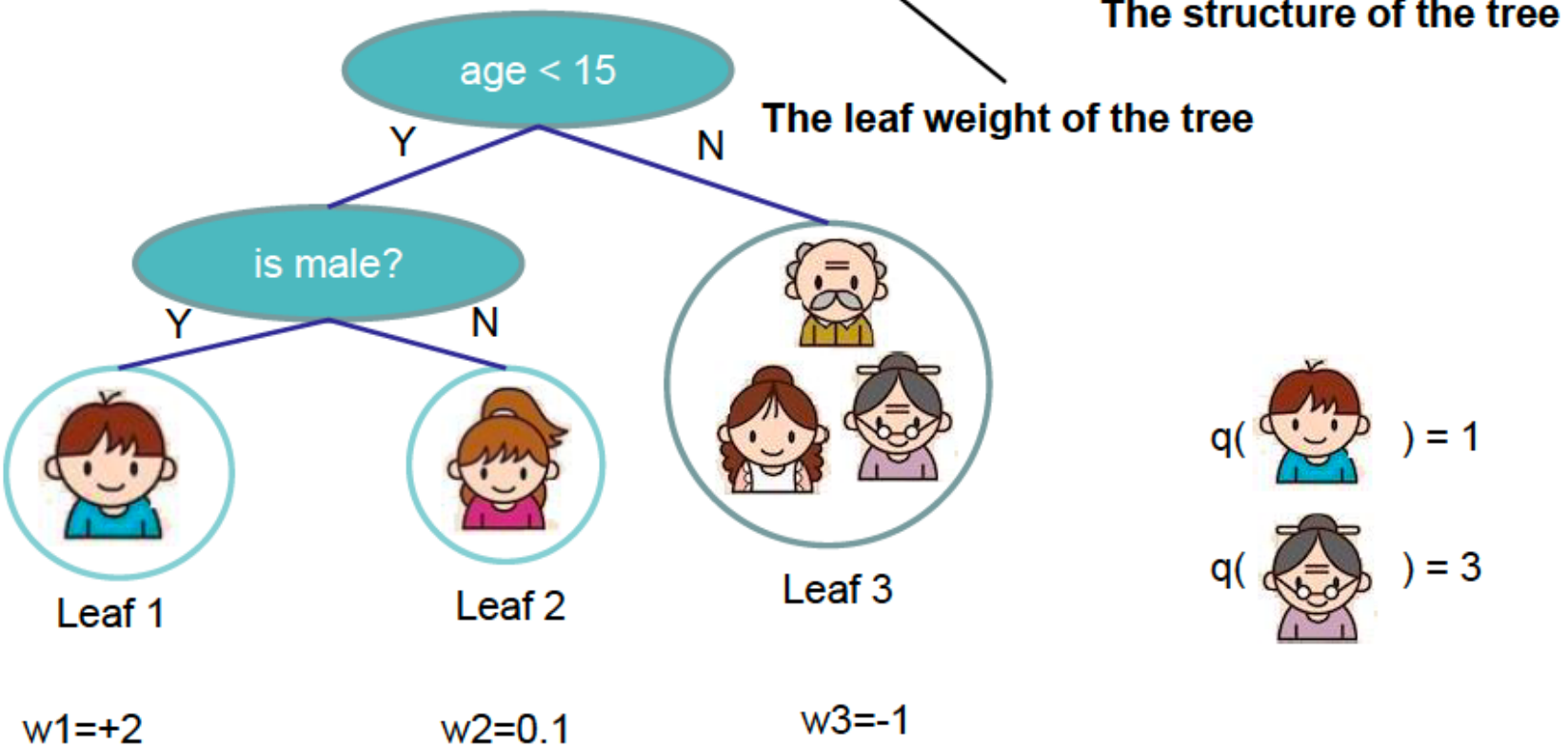
Refine the definition of tree

- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf.

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

The structure of the tree

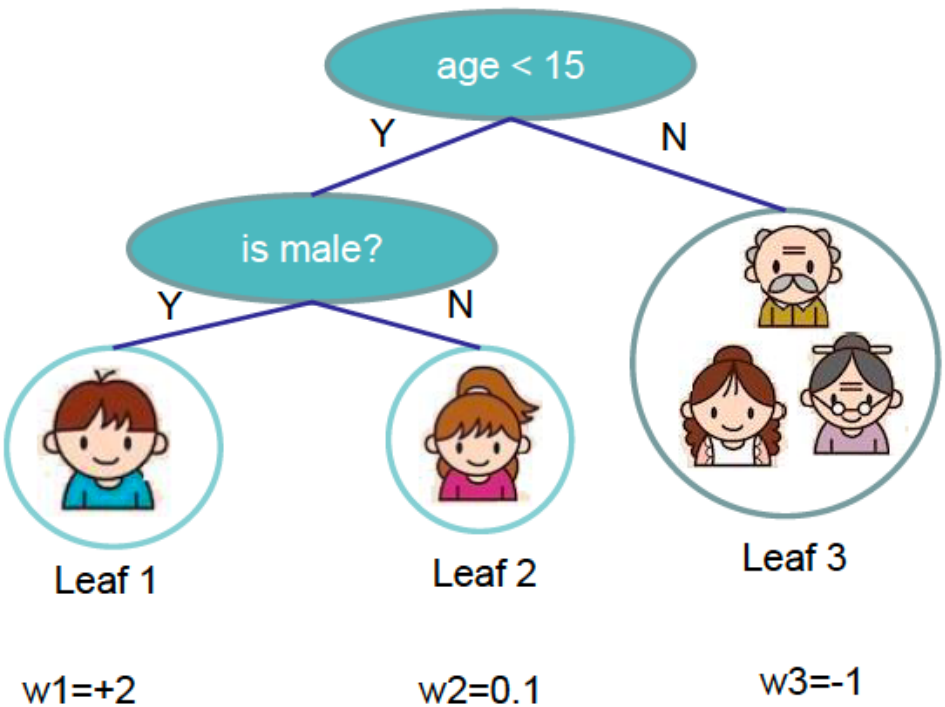
The leaf weight of the tree



Define Complexity of a Tree (cont')

- Define complexity as (this is not the only possible definition)
  - Number of leaves
  - L2 norm of leaf scores

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



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Our New Goal

- Objective, with constants removed

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

- Define the instance set in leaf j as $I_j = \{i|q(x_i) = j\}$
  - Regroup the objective by leaf

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^{n}[g_i \omega_{q(x_i)} + \frac{1}{2}h_i \omega^2_{q(x_i)}] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i \in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)\omega_j^2] + \gamma T \end{aligned}$$

- This is sum of T independent quadratic function

The Structure Score

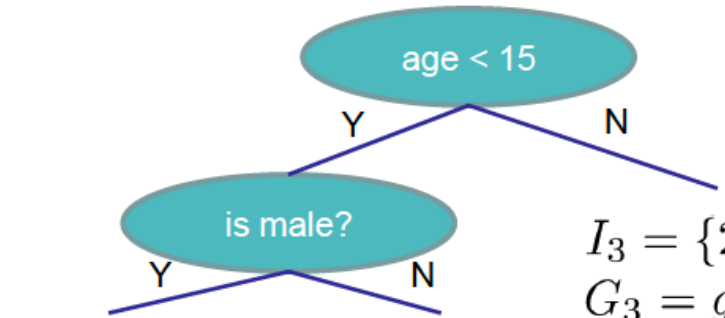
- Two facts about single variable quadratic function
  $argmin_x Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}$, $H > 0$ $\min_x Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$
- Let us define $G_j = \sum_{i \in I_j} g_i$, $H_j = \sum_{i \in I_j} h_i$

$$
\begin{aligned}
Obj^{(t)} &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)\omega_j^2] + \gamma T \\
&= \sum_{j=1}^{T} [G_j \omega_j + \frac{1}{2}(H_j + \lambda)\omega_j^2] + \gamma T
\end{aligned}
$$

- Assume the structure of tree ($q(x)$) is fixed, the optimal weight in each leaf, and the resulting objective value are
  $\omega_j^* = -\frac{G_j}{H_j + \lambda}$, $Obj = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$

The Structure Score Calculation

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

Searching Algorithm for Single Tree

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.
  $Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$
- Find the best tree structure, and use the optimal leaf weight
  $\omega_j^* = -\frac{G_j}{H_j+\lambda}$
- But... there can be infinite possible tree structures

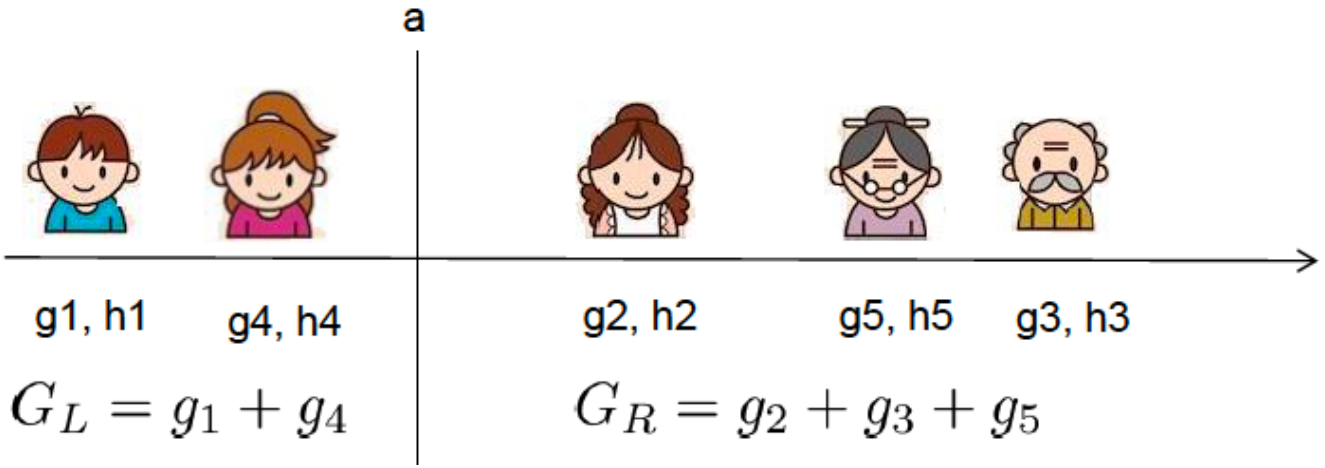
Greedy Learning of the Tree

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
  Gain $= \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$
- Remaining question: how do we find the best split?

Efficient Finding of the Best Split

- What is the gain of a split rule $x_j < a$? Say x_j is age



- All we need is sum of g and h in each side, and calculate $\text{Gain} = \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$
- Left to right linear scan over sorted instance is enough to decide the best split along the feature

An Algorithm for Split Finding

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features
- Time Complexity growing a tree of depth m
  - It is $\mathcal{O}(ndm \log n)$: or each level, need $\mathcal{O}(n \log n)$ time to sort There are d features, and we need to do it for m level
  - This can be further optimized (e.g. use approximation or caching the sorted features)
  - Can scale to very large dataset

What about Categorical Variables?

- Some tree learning algorithm handles categorical variable and continuous variable separately
  - We can easily use the scoring formula we derived to score split based on categorical variables.
- Actually it is not necessary to handle categorical separately.
  - We can encode the categorical variables into numerical vector using one-hot encoding. Allocate a #categorical length vector
    $$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & otherwise \end{cases}$$
  - The vector will be sparse if there are lots of categories, the learning algorithm is preferred to handle sparse data

Pruning and Regularization

- Recall the gain of split, it can be negative!
  Gain $= \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$
  - When the training loss reduction is smaller than regularization
  - Trade-off between simplicity and predictivness
- Pre-stopping
  - Stop split if the best split have negative gain
  - But maybe a split can benefit future splits..
- Post-Prunning
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate
  $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$, $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$
- Use the statistics to greedily grow a tree $f_t(x)$
  $Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$
- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \epsilon f_t(x_i)$
  - 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

Question

Try to compare the training speed of Bagging of decision trees, Random Forest and XGBoost.

Kaggle Winning Solution

To get a higher rank, one needs to push the limit of

1. Feature Engineering
2. Parameter Tuning
3. Model Ensemble

The winning solution in the Otto Competition is an excellent example. They used a 3-layer ensemble learning model, including

1. 33 models on top of the original data
2. XGBoost, neural network and adaboost on 33 predictions from the models and 8 engineered features
3. Weighted average of the 3 prediction from the second step

What you need to know

- Classification tree
- Model averaging techniques
  - Bagging
  - Random forests
  - Boosting

Outline

1. Introduction

2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost

3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Background

- Toutiao's platform has accumulated **580 million** users and **300 thousand** professional freelance writers.
- Toutiao Q&A is an up-and-coming mobile social platform, built upon 530 million Toutiao users. It is striving to match information with the right people, finding the best respondent to the questions, and the best readers to the answers.



Background

- How to match questions with interested users' expertise?
- Better recommendation strategy leads to more efficient Q&A system!
- Task: Predict whether an expert will answer a question.
- **HP:** https://biendata.com/competition/bytecup2016/

Task

- Participating teams will use data provided by Toutiao Q&A to conduct analysis on specific questions. Each data record includes expert tags, question data and question distribution data. Given certain questions, participants need to forecast which experts are more likely to answer which questions.
- In terms of evaluation, given a certain question, we will rank the experts based on the forecasted probability, and evaluate the NDCG@5 and NDCG@10 of ranking results. The final evaluation formula is:

$$Score = NDCG@5 * 0.5 + NDCG@10 * 0.5. \quad (12)$$

NDCG Score

- CG(cumulative gain) is a measure to evaluate the quality of listed experts.

$$CG@p = \sum_{i=1}^{p} rel_i \tag{13}$$

- DCG(Discounted CG) is a measure considering the rank of listed experts.

$$DCG@p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} \tag{14}$$

- nDCG(normalized DCG) is a measure that normalizes DCG to a range of $(0, 1)$.

$$nDCG@p = \frac{DCG@p}{\sum_{i=1}^{p} \frac{1}{log_2(i+1)}} \tag{15}$$

- rel_i denotes the relativeness of the i^{th} listed experts to the question. $rel_i = 1$ if expert i answers the question, otherwise $rel_i = 0$.

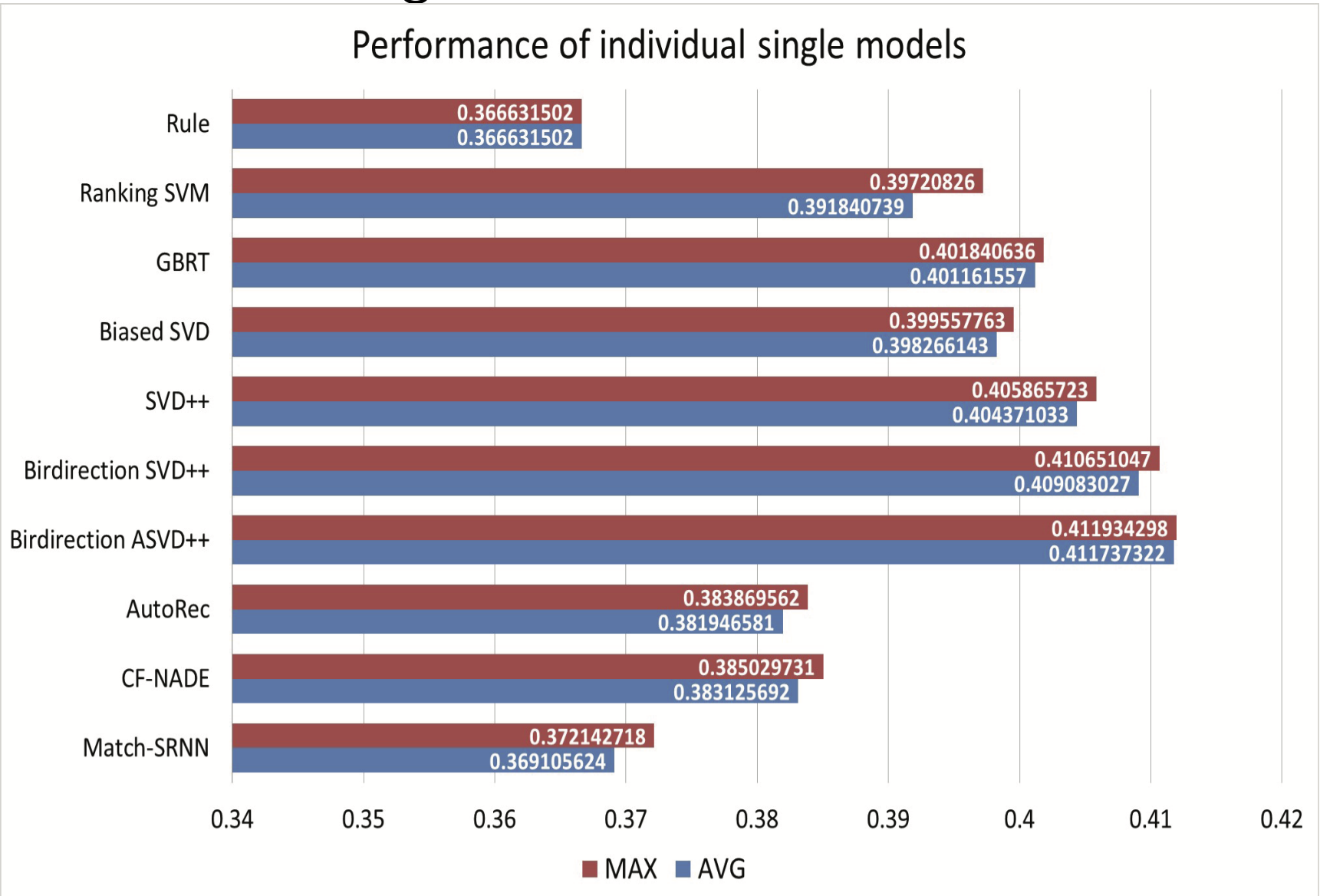
Outline

1. Introduction

2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost

3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Single Model Performance



- Best single model achieves < 0.42 *Score*.

LeaderBoard

Final Leaderboard - 2016 BYTECUP

If you see anyone use multiple accounts, please let us know.

#	Team Name	Score	Final Entries
1	BrickMover	0.50812	17
2	天穹	0.50307	7
3	西电战队	0.49905	17
4	charles	0.49832	8
5	不得平视本王	0.49315	2
6	MLCLASS_Virtuoso	0.49231	8
7	USTC	0.49003	13
8	kuhung	0.48978	2
9	Fengari	0.48830	1
10	topgun	0.48797	18
11	hesu	0.48779	4
12	xiaonezha	0.48770	1
13	Three kids	0.48770	1
14	beginner	0.48749	2
15	aida	0.48717	1
16	MLCLASS_PHOENIX	0.48714	7
17	MLCLASS_OM	0.48710	15
18	MLCLASS_BayesCase	0.48704	11
19	DeepLearner	0.48693	1
20	Dplyr	0.48686	2

LeaderBoard

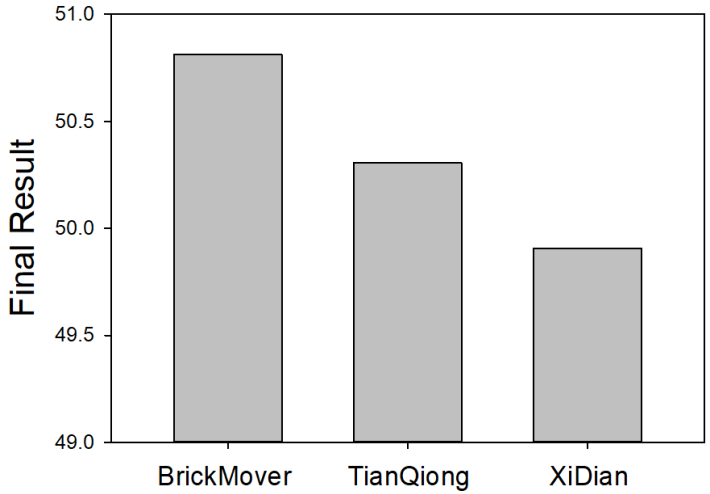
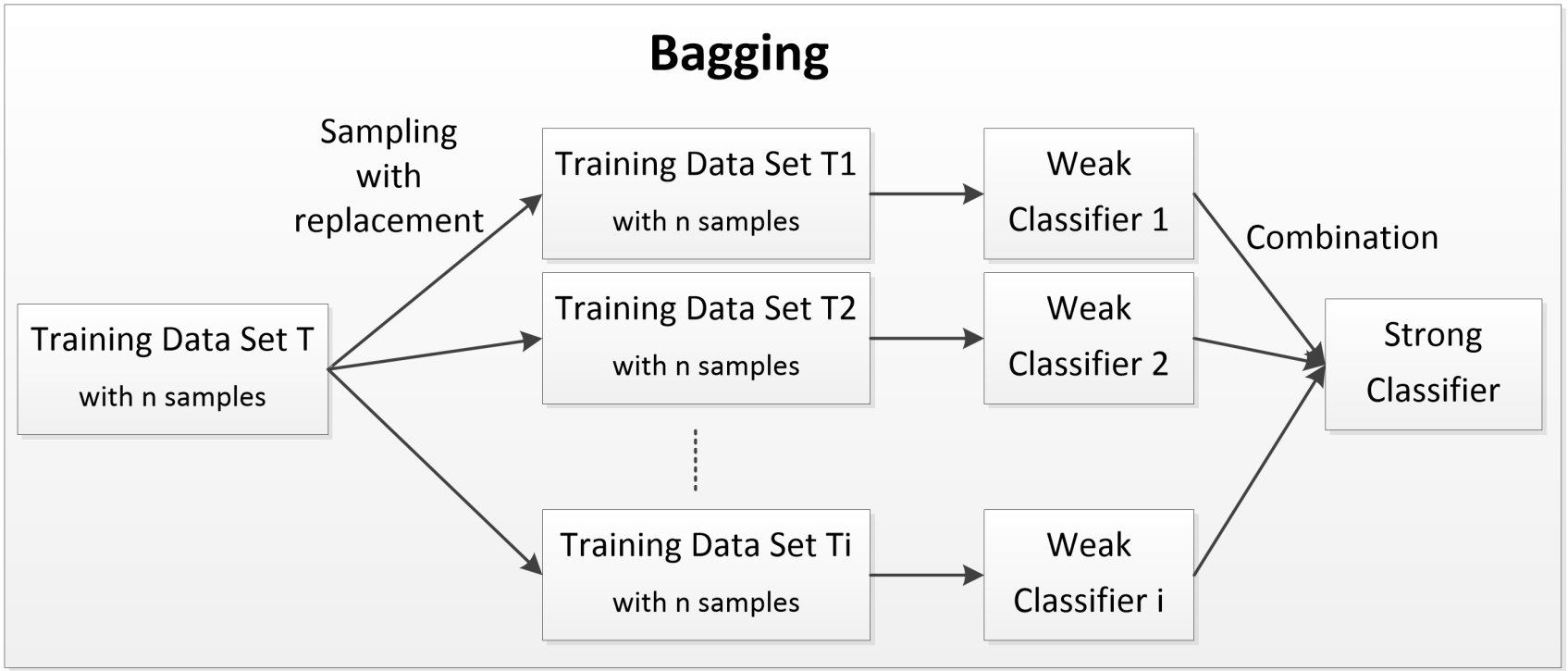


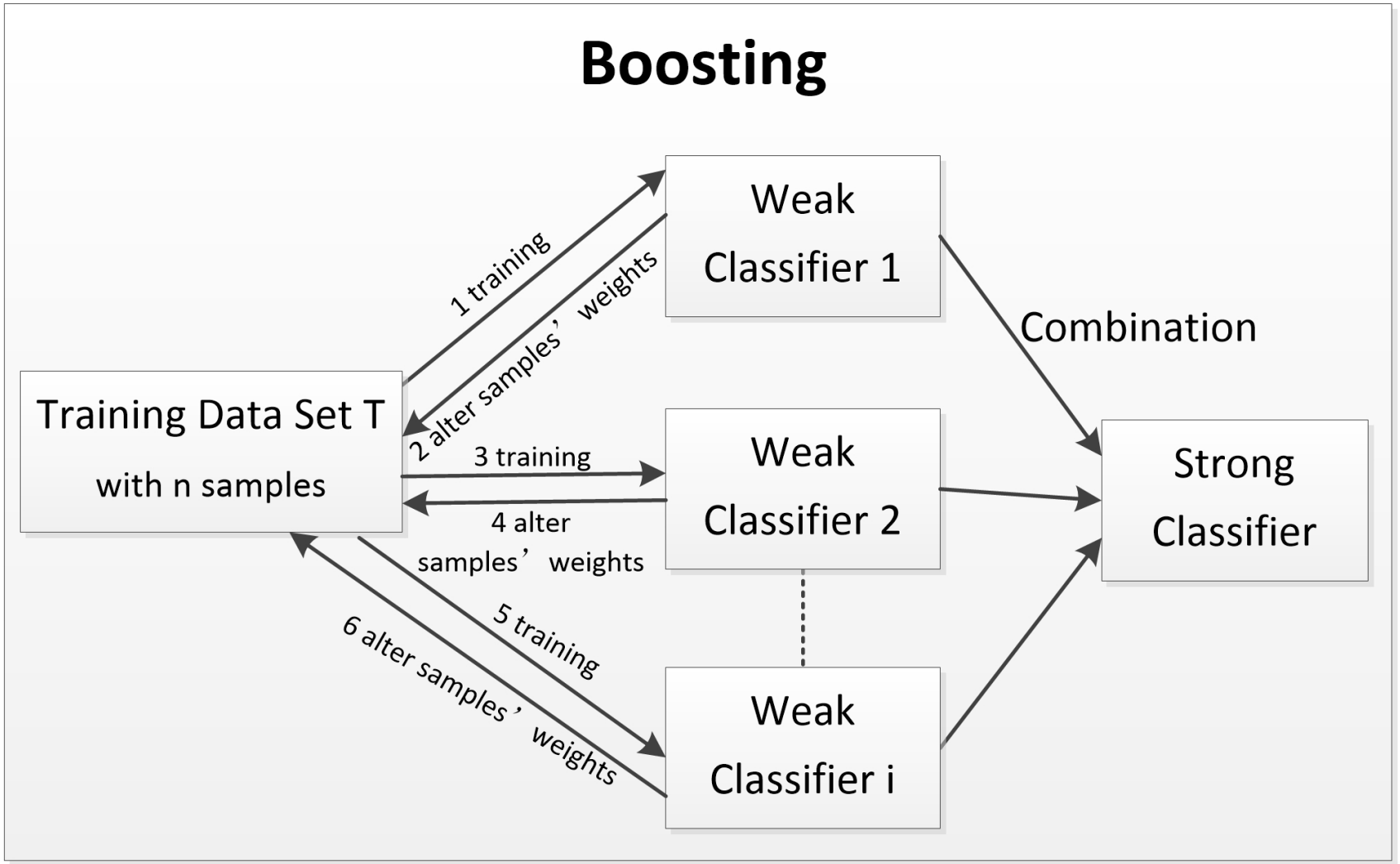
Figure 1: *Brickmover* get $Score = 0.508$ and was awarded \$5000. *TianQiong* and *Xidian* get $Score = 0.503, 0.499$ and were awarded \$3000, \$2000 respectively.

- More than 100 teams (out of 129) achieve $Score > 0.42$.
- You need $Score > 0.4868$ to rank in top 20.
- Best single model only gives you $Score = 0.412$.
- Most candidates use various ensemble techiques.

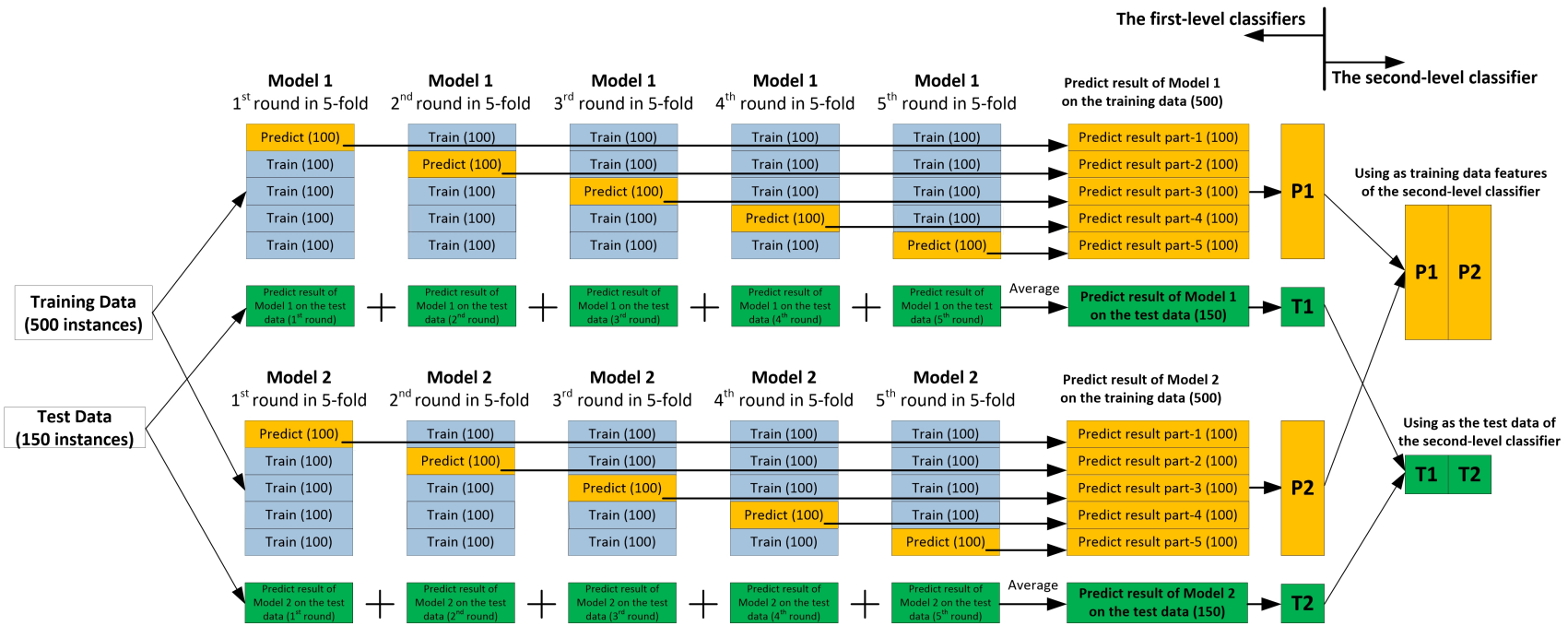
Bagging


Bagging
Sampling with replacement
Training Data Set T with n samples
Training Data Set T1 with n samples
Training Data Set T2 with n samples
Training Data Set Ti with n samples
Weak Classifier 1
Weak Classifier 2
Weak Classifier i
Combination
Strong Classifier

Boosting


Boosting
Training Data Set T
with n samples
Weak
Classifier 1
Weak
Classifier 2
Weak
Classifier i
Combination
Strong
Classifier
1 training
2 alter samples' weights
3 training
4 alter
samples' weights
5 training
6 alter samples' weights

Stacking


The first-level classifiers
The second-level classifier
Model 1
1st round in 5-fold
Model 1
2nd round in 5-fold
Model 1
3rd round in 5-fold
Model 1
4th round in 5-fold
Model 1
5th round in 5-fold
Predict result of Model 1 on the training data (500)
Predict (100)
Train (100)
Predict result part-1 (100)
Predict result part-2 (100)
Predict result part-3 (100)
Predict result part-4 (100)
Predict result part-5 (100)
P1
Using as training data features of the second-level classifier
Training Data (500 instances)
Predict result of Model 1 on the test data (1st round)
Predict result of Model 1 on the test data (2nd round)
Predict result of Model 1 on the test data (3rd round)
Predict result of Model 1 on the test data (4th round)
Predict result of Model 1 on the test data (5th round)
Average
Predict result of Model 1 on the test data (150)
T1
P1
P2
Model 2
1st round in 5-fold
Model 2
2nd round in 5-fold
Model 2
3rd round in 5-fold
Model 2
4th round in 5-fold
Model 2
5th round in 5-fold
Predict result of Model 2 on the training data (500)
P2
Test Data (150 instances)
Using as the test data of the second-level classifier
Predict result of Model 2 on the test data (1st round)
Predict result of Model 2 on the test data (2nd round)
Predict result of Model 2 on the test data (3rd round)
Predict result of Model 2 on the test data (4th round)
Predict result of Model 2 on the test data (5th round)
Average
Predict result of Model 2 on the test data (150)
T2
T1
T2

Outline

1. Introduction

2. Algorithms
   - Bagging
   - Random Forests
   - Boosting
   - Stacking
   - XGBoost

3. Nearby Application–Toutiao Byte Cup 2016
   - Methods
   - Best Solutions

Xidian Solution

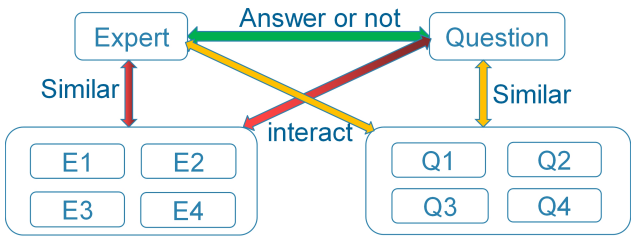


Figure 2: Xidian-CF

Question	Expert	Label
qid1	uid1	0
qid1	uid2	1
qid2	uid1	1

Design features

qid		uid		uq1	uq2	qu1	qu2	label
1	0	1	0	1/2	1/2	1/2	1/2	0
1	0	0	1	1/1	0	1/2	1/2	1
0	1	1	0	1/2	1/2	1/1	0	1

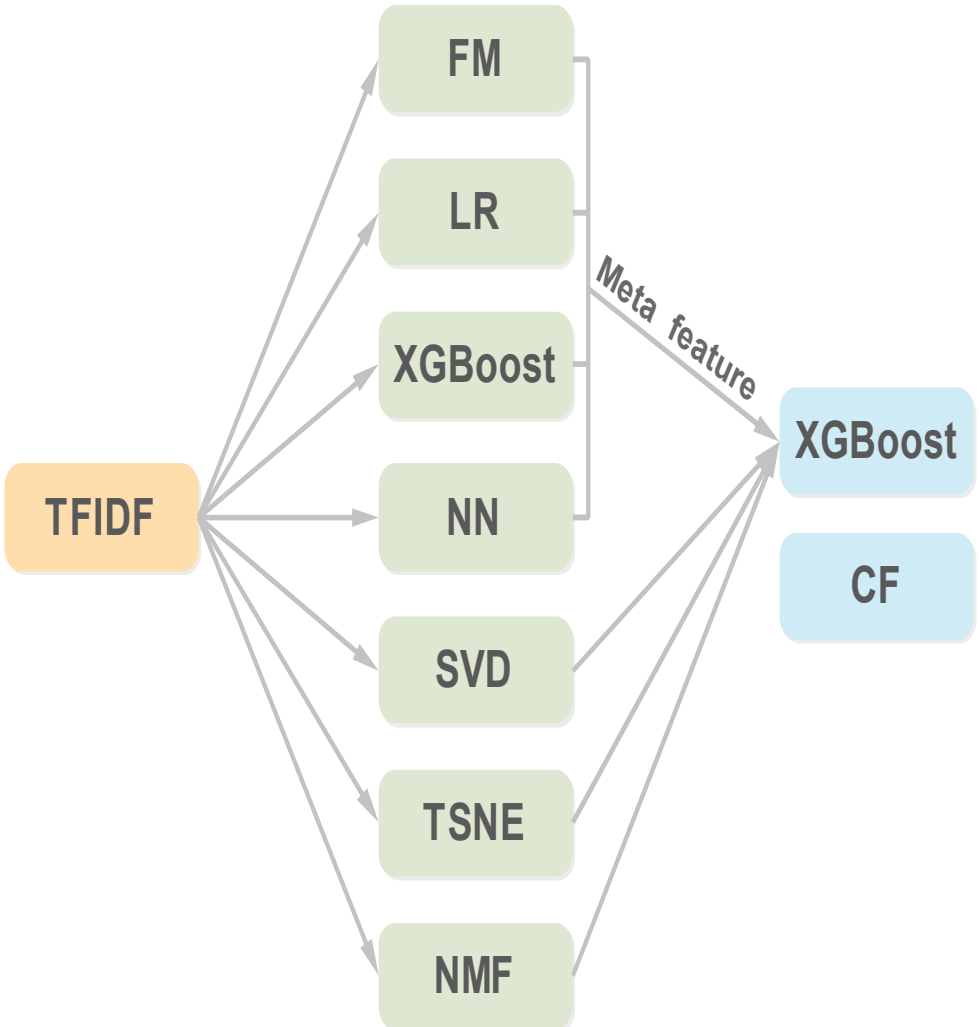
Training

FM Algorithm

Figure 3: Xidian-FM

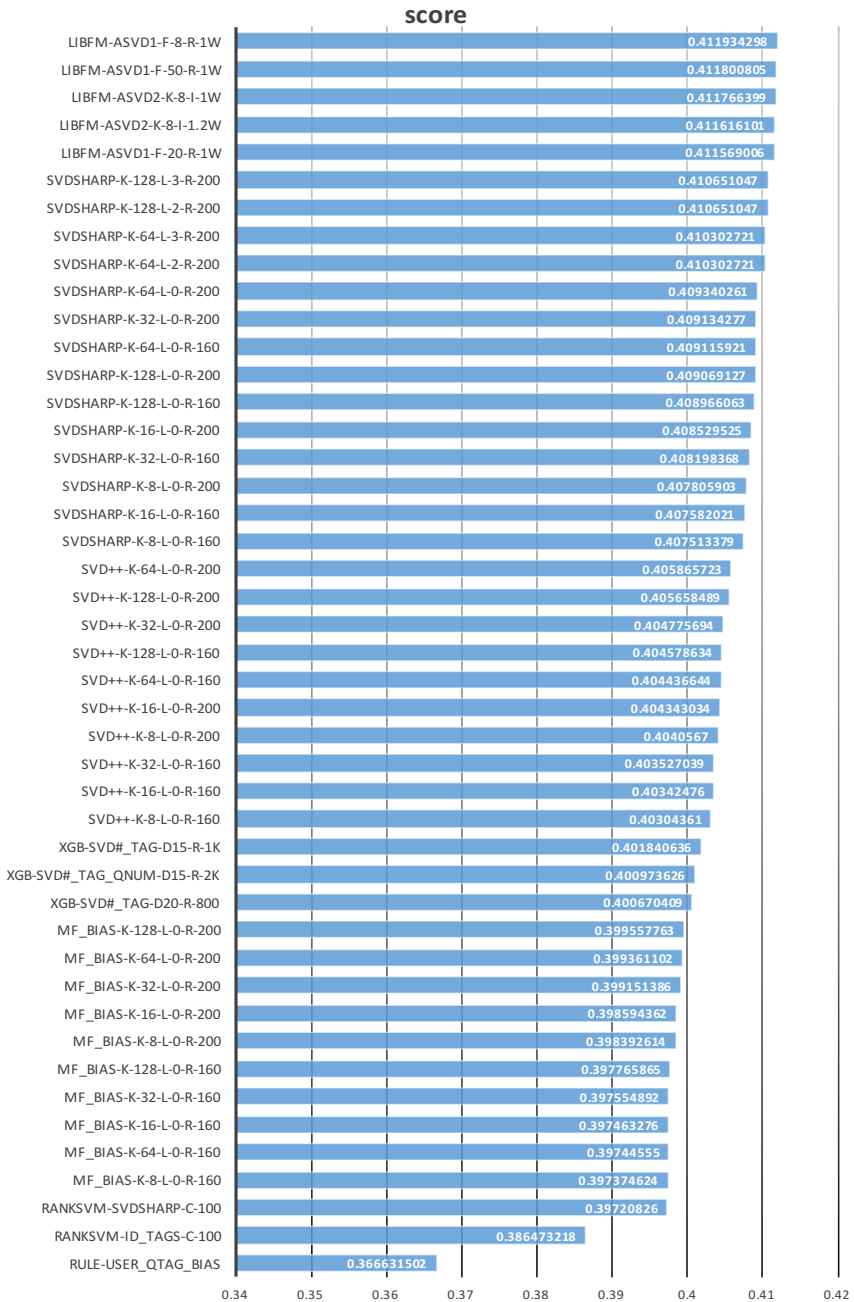
- Design Problem-specific CF and FM model.
- Use linear weighted sum of the two models as output.
- Final $Score = 0.499$.

TianQiong Solution



- A complicated stacking model.
- Stack 10 different methods at different levels.
- Final *Score* = 0.503

Brickmover Solution



- Train 45 different models using different methods/same method with different training hyperparameters.
- These weak single models have performance ranging from $Score = 0.366$ to $Score = 0.412$.
- Linearly combine all of them.
- Final $Score = 0.508$

Discussion

Table 7 Ensemble models used by the top 5 teams.

Method / Team	Details of the ensemble model	Final results	Compare with Team-1
Team-1	Linearly combine all models in Fig. 3	0.50812	0
Team-2	Use Stacking strategy illustrated as Fig. 5	0.50307	−1%
Team-3	FM+CF *	0.49905	−1.82%
Team-4	MF+CF	0.49231	−3.21%
Team-5	FM+RFM+(FM+RFM)+MF+SVD+(SVD++)	0.49003	−3.69%

* FM+CF represents the linear weighted sum of FM and CF.

- All top 5 teams use ensemble methods.
- Different team uses different base classifiers and ensemble methods.
- As Toutiao has 580 million users, every 0.1% improvement is significantly useful.
- Ensembling is really useful in real world application.

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn