Lecture 6

Trees and ensembles

Machine Learning

Andrey Filchenkov

11.10.2019

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

- The presentation is prepared with materials of the K.V. Vorontsov's course "Machine Leaning".
- Slides are available online: **bit.ly/2lynu3L**

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Concepts and rules

Concept is a predicate on an object set X:

$$\varphi: X \to \{0,1\}.$$

A concept **covers** an object x, if $\varphi(x) = 1$.

Rule is a logical predicate which covers many objects from one class and few objects from other classes, and which can be simply interpreted.

Example (from the Russian language): *If a word is an adverb and it ends with a hissing sound ("ж", "ч" or "ш"), you should end it with "ь".*

Examples: "вска**чь**", "насте**жь**".

Exceptions: "уж", "замуж", "невтерпеж".

Interpretable concepts

Origins in knowledge discovery in databases.

A concept φ **can be interpreted** if it

1) is formulated in natural language;
2) depends on a small number of feature (1–7).

Informative concepts

A concept φ is **informative** for a class c, if

$$p(\varphi) = |\{x_i | \varphi(x_i) = 1, y_i = c\}| \rightarrow \max;$$
$$n(\varphi) = |\{x_i | \varphi(x_i) = 1, y_i \neq c\}| \rightarrow \min.$$

Can be reformulated in a probabilistic sense.

$p(\varphi)$ is True Positive;

$n(\varphi)$ is False Positive;

$p(\varphi) + n(\varphi)$ is coverage.

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Convolution choice problem

It is not obvious, how to convolute these two parameters:

- Precision:

$$\frac{p}{p+n} \to \max;$$

- Accuracy

$$p - n \to \max;$$

- Linear cost accuracy:

$$p - Cn \to \max;$$

- Relative accuracy:

$$\frac{p}{P} - \frac{n}{N} \to \max.$$

Convolution choice comparison

Compare with $P = N = 100$:

p	n	$p-n$	$p-5n$	$\frac{p}{P}-\frac{n}{N}$	$\frac{p}{n+1}$	$\frac{p}{n+p}$	I_c	IGain_c	$\sqrt{p}-\sqrt{n}$
50	0	**50**	50	0.25	50	1	22.65	23.70	7.07
100	50	**50**	-150	0	1.96	0.67	2.33	1.98	2.93
50	9	41	**5**	0.16	**5**	**0.85**	7.87	7.94	4.07
5	0	5	**5**	0.03	**5**	**1**	2.04	3.04	2.24
100	0	**100**	100	**0.5**	100	1	52.18	53.32	10.0
140	20	**120**	40	**0.5**	6.67	0.88	37.09	37.03	7.36

ε, δ-rule

$E_c(\varphi, T^\ell) = \frac{n_c(\varphi)}{p_c(\varphi)+n_c(\varphi)}$ is a share of falsely covered objects.

$D_c(\varphi, T^\ell) = \frac{p_c(\varphi)}{\ell}$ is a share of positive objects among the covered objects.

$\varphi(x)$ is **ε, δ-rule** (for class c), if $E_c(\varphi, T^\ell) \leq \varepsilon$ and $D_c(\varphi, T^\ell) \geq \delta$.

If $n_c(\varphi) = 0$, then the rule is **exact**.

Statistical rule

Assumption: the sample is simple.

Probability of pair (p, n) is described with hyper-geometric distribution:

$$\mathcal{H}_{P,N}(p, n) = \frac{C_P^p C_N^n}{C_{P+N}^{p+n}}.$$

Comprehension of predicate φ with sample T^ℓ:

$$I_c(\varphi, T^\ell) = -\ln \mathcal{H}_{P_c,N_c}\big(p_c(\varphi), n_c(\varphi)\big).$$

$\varphi(x)$ is **statistical rule** (for class c), if

$$I_c(\varphi, T^\ell) \geq \alpha$$

with α being high enough (**Fisher exact test**).

Entropy-based rule

Entropy of two outcomes:

$$H(q_0, q_1) = -q_0 \log_2 q_0 - q_1 \log_2 q_1 .$$

Entropy of the sample:

$$\widehat{H}(P, N) = H\left(\frac{P}{P+N}, \frac{N}{P+N}\right).$$

$$\widehat{H}_{\varphi}(P, N, p, n) =$$

$$= \frac{p+n}{P+N}\widehat{H}(p, n) + \frac{P+N-p-n}{P+N}\widehat{H}(P-p, N-n).$$

$$\mathrm{IGain}_c\left(\varphi, T^{\ell}\right) = \widehat{H}(P, N) - \widehat{H}_{\varphi}(P, N, p, n).$$

$\varphi(x)$ is **entropy-based rule** (for class c), if $\mathrm{IGain}_c\left(\varphi, T^{\ell}\right) \geq G_0$ with a certain G_0 being high enough.

Good criteria (convolutions)

- IGain_c, I_c

  Theorem: $\lim\limits_{\ell\to\infty} \mathrm{IGain}_c(\varphi, T^\ell) = \frac{1}{\ell \ln 2} I_c(\varphi, T^\ell)$.
- Boosting criterion:
$$\sqrt{p} - \sqrt{n} \to \max.$$
- Normalized boosting criterion:
$$\sqrt{\frac{p}{P}} - \sqrt{\frac{n}{N}} \to \max.$$

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Rule definition and examples (1/2)

Rule is an interpretable highly informative single-class classifiers with refusals.

Examples

1. **Conjunction of boundaries** (terms):

$$R(x) = \bigwedge_{j \in J} [a_j \le f(x_j) \le b_j].$$

Rule definition and examples (2/2)

2. **Syndrome** is at least d terms of J are true:

$$R(x) = \left[\sum_{j\in J}[a_j \le f(x_j) \le b_j] \ge d\right],$$

(when $d = |J|$, it is conjunction, when $d = 1$, it is disjunction).

3. **Half-plane**:

$$R(x) = \left[\sum_{j\in J} w_j f_j(x) \ge w_0\right].$$

4. **Ball** is threshold similarity function:

$$R(x) = [r(x, x_0) \le r_0].$$

Where to get concepts and how to choose rules

Concepts can be:

- created by yourself;
- learnt;
- given from experts.

Rules can be learnt with

- optimization problem solvers;
- heuristic methods;
- special machine learning algorithms.

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Decision trees

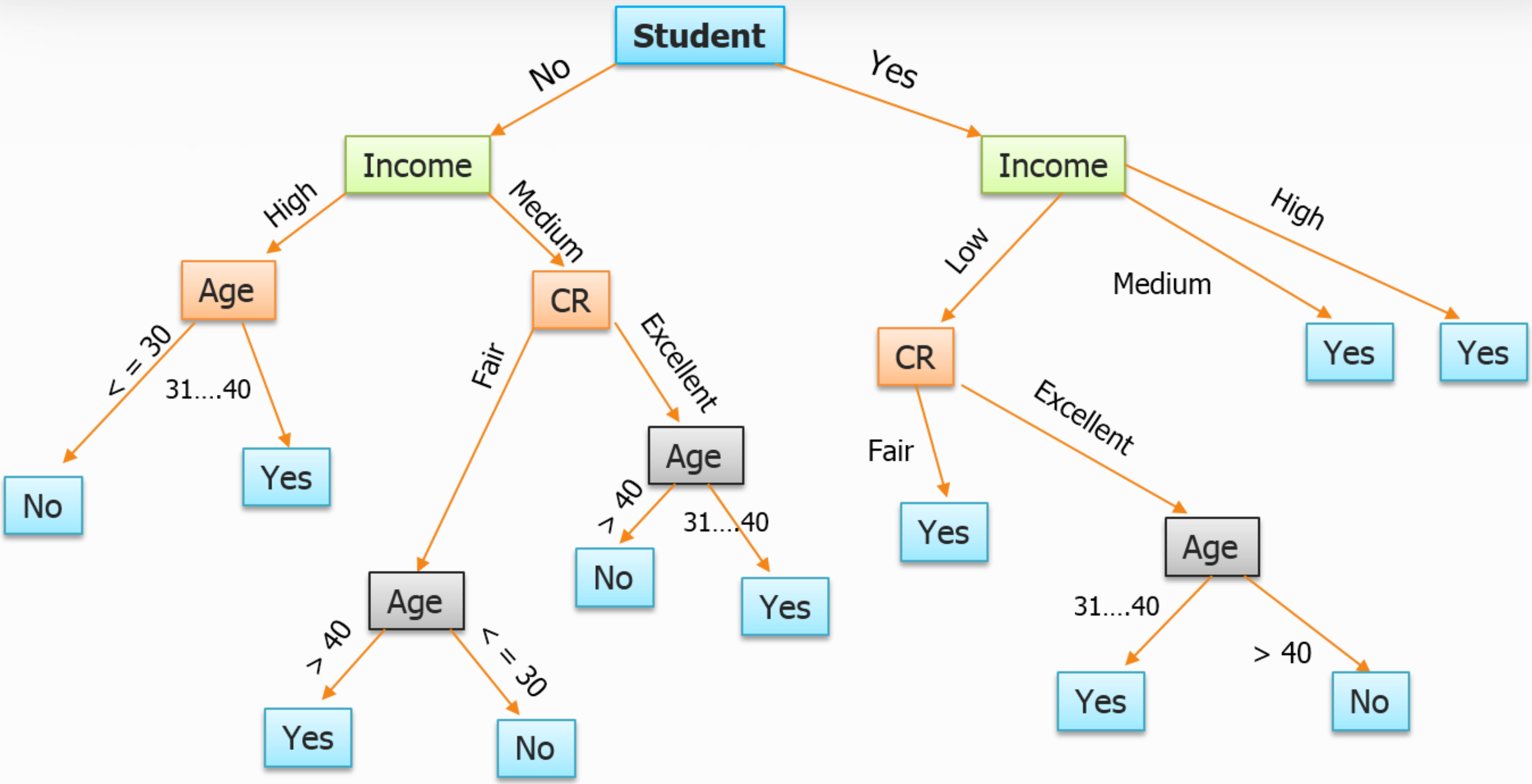
Decision tree is a classifier and a regression algorithm.

Nodes contain splitting rules (questions).

Each edge is a possible answer to its node question.

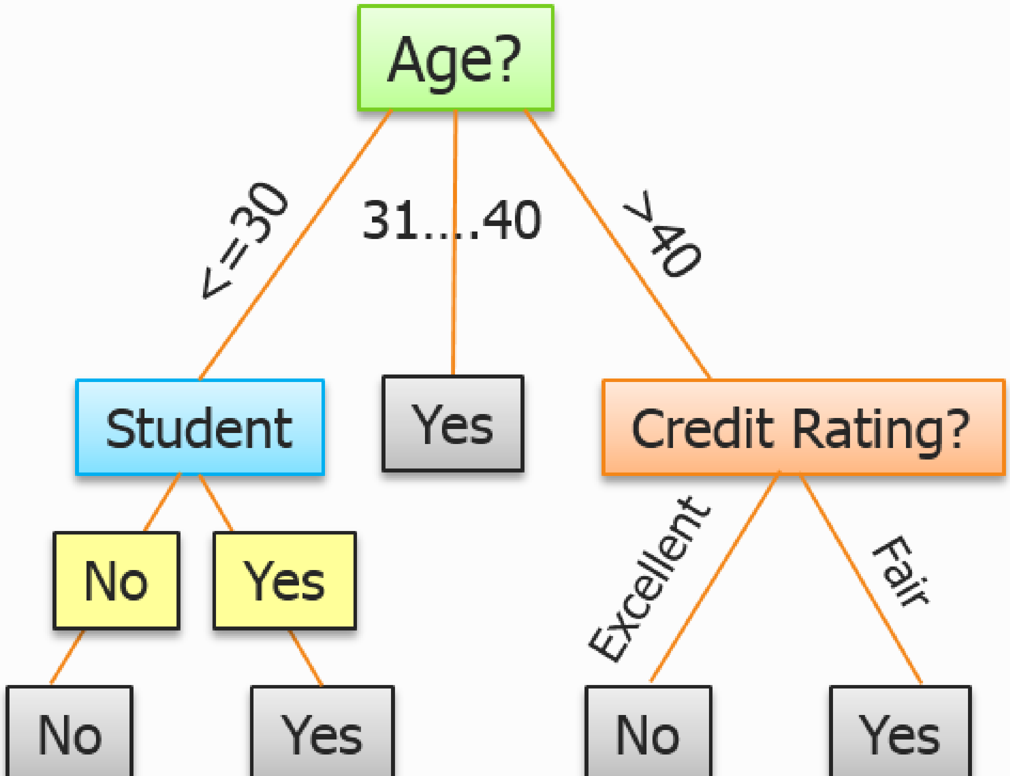
Leafs contain decisions (a class for classification problem and a number for regression problem).

Decision tree example (1/2)



Decision tree example (2/2)

The same classification can be achieved in a much more simpler way



General scheme

With splitting rules space $\mathcal{B}$ and split quality measure Φ.

1. Sent S to the root.

2. On each step process sample S.

2.1. If S contains objects from a single class c, create a leaf of the class c and stop.

2.2. Else choose a splitting rule $\mathscr{b} \in \mathcal{B}$, the most informative with respect to Φ, and split the sample to $S_1, \dots, S_k$.

2.3. If stop-criterion is true, then return the most popular class in S, otherwise create k children with samples S_i.

3. Prune the resulting tree.

Four main questions

How to choose splitting rules?

How to choose split quality measure (splitting criterion)?

How to choose a stop-criterion?

How to prune the tree?

Selecting splitting rules family

Can be any family of classifiers.

- In most of the cases, it is single-feature-based rules like:

$$f_i(x) > m_i;$$
$$f_i(x) = d_i.$$

- Sometimes, it can be a combination.

Selection of feature-based rules

There are $\ell - 1$ options to split the sample.

- Check each and pick the most informative.
- Join diapasons of values.
- Skip small diapasons.

This is how you can synthesize a rule for each feature.

Sample splitting

- If a sample is split each time into 2 ($k = 2$), then $\mathcal{B}$ is a family of binary rules, tree is binary.
- If a feature is categorical, several edges can be added.
- If a feature is real, discretization / binarization is applied.

On each step, the number of edges can differ, but usually k is fixed.

Selecting split quality measure

Split quality measure Φ can be sometimes represented as:

$$\Phi(S) = \phi(S) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} \phi(S_i).$$

The most popular:

- $\phi_h(S)$ is entropy, $\Phi_h(S)$ is **IGain**;
- $\phi_g(S) = 1 - \sum_{i=1}^{m} p_i^2$, where p_i is probability (frequency) of ith class in sample S is **Gini index**. $\Phi_h(S)$ is **GiniGain**.

Many other are used, for instance

GainRatio = IGain(S) / Enthropy(S).

Split quality measure usually does not effect on tree efficacy.

Stop-criteria

The most popular stop-criteria:

- one of classes is empty after splitting
- $\Phi(S)$ is lower than a certain threshold
- $|S|$ is lower than a certain threshold
- tree height is higher than a certain threshold

Prepruning

Another type of stop-criteria.

Stop growing the tree when there is no statistically significant association between any feature and the class at a particular node

Typically, chi-squared test is used

Pruning

Premises: just first node impact on performance; decision trees tend to be overfitted.

Main idea: to cut lower branches.

Pruning is processing of created trees, when simplification operators are applied consequently if they improve performance with respect to a certain criterion (reduction number of errors, for example).

Pruning algorithm scheme

Split sample to train and control in proportion 2:1.

For each tree node apply a simplification operator, which is the best in terms of performance criterion:

1) Do not change anything;
2) Replace node with its child (for each child);
3) Replace node with a leaf (for each class).

Examples

ID3 (Quinlan, 1986):

IGain; only $\Phi(S) < 0$; no pruning.

C4.5 (Quinlan, 1993):

GainRatio; pruning.

CART (Breinman, 1984):

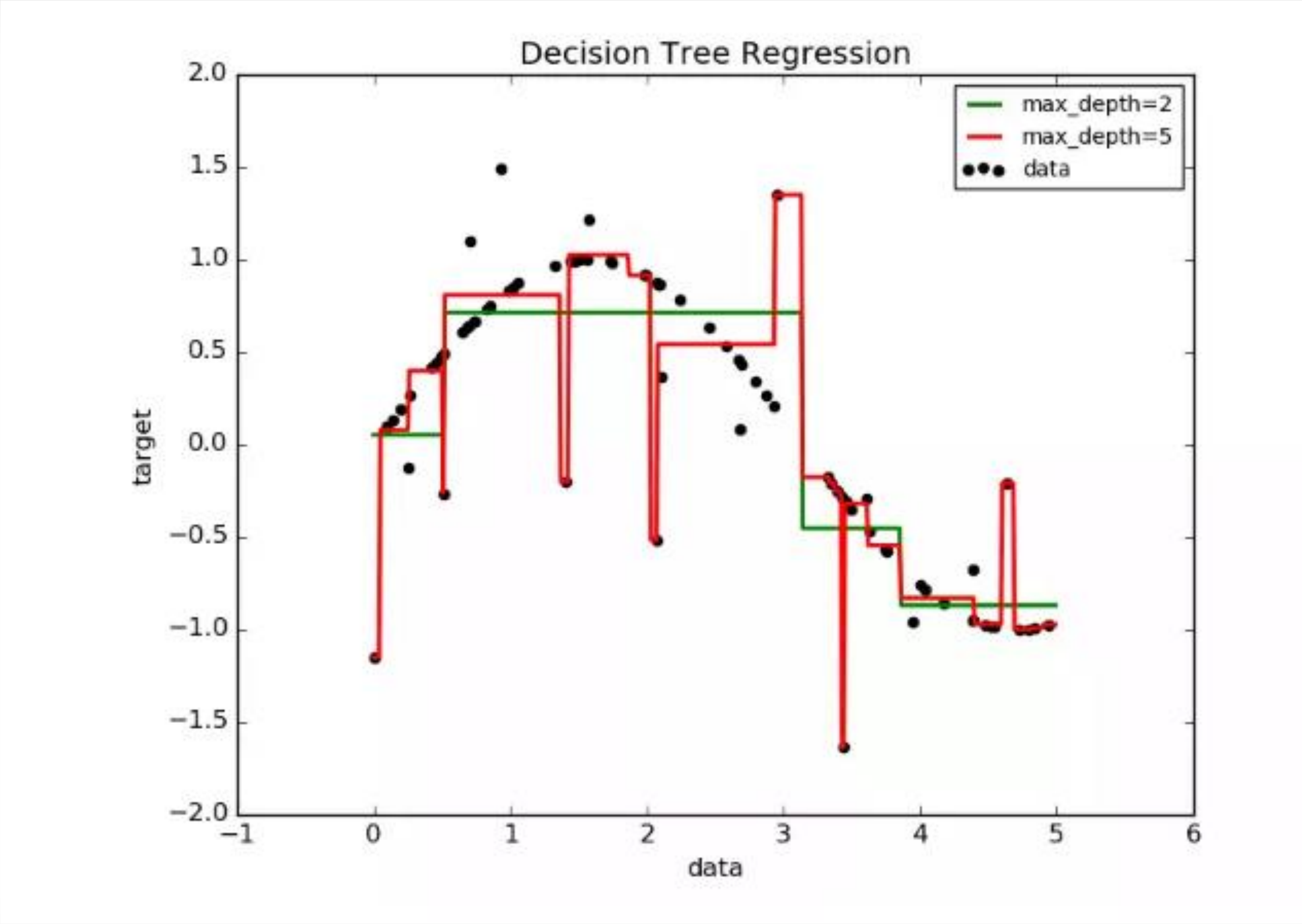
binary; GiniGain; pruning. Can solve regression (values in leafs).

Regression trees

Regression trees are used for solving regression.

They work in the same way as decision trees, but each leaf return average target value of the objects it has instead of the most popular class.

Regression tree example



Model trees

Model trees are used for solving regression.

They work in the same way as decision and regression trees, but each leaf return some model hyperparameters.

Typically, linear model is used.

Trees discussion

Advantages:

- easily understandable and interpretable;
- learning is quick;
- can work with different data type;
- perform feature selection.

Disadvantage:

- sensitive to noise;
- easily get overfitted.

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Weak and strong learnability

Weak learnability means that one can find an algorithm in polynomial time, performance of which would be more than 0.5.

Strong learnability means that one can find an algorithm in polynomial time, performance of which would be any high.

What is true: weak or strong learnability?

Weak and strong learnability

Weak learnability means that one can find an algorithm in polynomial time, performance of which would be more than 0.5.

Strong learnability means that one can find an algorithm in polynomial time, performance of which would be any high.

Theorem (Schapire, 1990)
Strong learnability is equivalent to weak learnability, because any model can be strengthen with algorithm composition.

Simple example

We have n algorithms with probabilities of correct classification answer $p_1, p_2, \ldots, p_n \approx p$. These probabilities are independent.

New algorithm will choose a class label with respect to the most preferable class within these algorithms.

Then probability of the correct classification answer is:

$$P_{vote} = p^n + np^{n-1}(1-p) + \frac{n(n-1)}{2} p^{n-2}(1-p)^2 + \cdots +$$

$$+C_n^{n/2} p^{n/2}(1-p)^{n/2}.$$

Problem formulation

Object set X, answer set Y.

Training sample $X^\ell = \{x_i\}_{i=1}^\ell$ with known labels $\{y_i\}_{i=1}^\ell$.

Family of basic algorithms

$$H = \{h(x,a): X \to R | a \in A\},$$

a is a parameter vector, which describes an algorithm, R is codomain (usually $\mathbb{R}$ or $\mathbb{R}^M$).

Problem: find (synthesize) a algorithm which is the most precise in forecasting label of object of X.

Composition of algorithms

Composition of N basic algorithms $h_1, \dots, h_N : X \to R$ is

$$H_T(x) = C(F(h_1(x), \dots, h_T(x))),$$

where $C: R \to Y$ is a **decision rule**, $F: R^T \to R$ is an **adjusting function**.

R is usually wider than Y.

Decision rule

Decision rule: $C\big(H(x)\big) \to Y$:

- for regression, $Y = \mathbb{R}$

$$C(H(x)) = H(x), \text{ or with a transformation.}$$

- for classification on k classes, $Y = \{1, \dots, k\}$

$$C(F(h_1(x), \dots, h_k(x))) = \operatorname{argmax}_{y \in Y} h_y(x)$$

- for binary classification, $Y = \{-1, +1\}$

$$C(H(x)) = \operatorname{sign}(H(x))$$

Usually this function is used:

$$L(H(x), y) = L(H(x)y)$$

Voting

The simplest example of the adjusting function is **voting**.

Two types of voting:

- majority voting (count votes)
- soft voting (count probabilities)

We can add weights for voters (better with soft voting).

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Boosting problem formulation

Let's synthesize an algorithm described as

$$H_T(x) = \sum_{t=1}^{T} b_t h(x, a_t),$$

where $b_t \in \mathbb{R}$ are the coefficients minimizing empirical risk

$$Q = \sum_{i}^{\ell} L(H_T(x_i), y_i) \to \min$$

for a loss function $L(H_T(x_i), y_i)$.

Gradient descent

It is hard to find an exact solution $\{(a_t, b_t)\}_{t=1}^{T}$.

We will develop function step by step

$$H_t(x) = H_{t-1}(x) + b_t h(x, a_t)$$

To do that, we estimate gradient of error function $Q^{(t)} = \sum_{i=1}^{\ell} L(H_t(x_i), y_i)$ incrementally.

This error function $Q^{(t)}$ is a vector with the length equal to the number of objects, ℓ:

$$Q^{(t)} = \left(Q_1^{(t)}, \dots, Q_\ell^{(t)}\right).$$

Gradient

Gradient (for ith element of $Q^{(t-1)}$):

$$\nabla Q_i^{(t-1)} = \frac{\delta Q_i^{(t-1)}}{\delta H_{t-1}(x_i)} = \frac{\delta\left(\sum_i^{\ell} L(H_{t-1}(x_i), y_i)\right)}{\delta H_{t-1}(x_i)} = \\ = \frac{\delta L(H_{t-1}(x_i), y_i)}{\delta H_{t-1}(x_i)}.$$

Thus, we will add

$$H_t(x) = H_{t-1}(x) - b_t \nabla Q^{(t-1)}.$$

Parameters selection

$$H_t(x) = H_{t-1}(x) - b_t \nabla Q^{(t-1)}$$

$$b_t = \operatorname{argmin}_b \sum_{i=1}^{\ell} L\left(H_{t-1}(x_i) - b \nabla Q^{(t-1)}, y_i\right).$$

Vector $\nabla Q^{(t-1)}$ is not a basic algorithm, so

$$a_t = \operatorname{argmin}_{a \in A} \sum_{i=1}^{\ell} L(h(x_i, a), \nabla Q^{(t-1)}) \equiv$$

$$\equiv \text{Learn}\left(\{x_i\}_{i=1}^{\ell}, \left\{\nabla Q_i^{(t-1)}\right\}_{i=1}^{\ell}\right).$$

We can find it by linear search

$$b_t = \operatorname{argmin}_b \sum_{i=1}^{\ell} L(H_{t-1}(x_i) - b h(x_i, a_t), y_i).$$

Generalized algorithm

Input: T^{ℓ}, N

$H_0(x) = \text{LEARN}\left(\{x_i\}_{i=1}^{\ell}, \{y_i\}_{i=1}^{\ell}\right)$

1. **for** $t = 1$ **to** T **do**
2. $\nabla Q^{(t-1)} = \left[\frac{\delta L(H_{t-1}, y_i)}{\delta H_{t-1}}(x_i)\right]_{i=1}^{\ell}$
3. $a_t = \text{LEARN}\left(\{x_i\}_{i=1}^{\ell}, \left\{\nabla Q_i^{(t)}\right\}_{i=1}^{\ell}\right)$
4. $b_t = \text{argmin}_b \sum_{i=1}^{\ell} L(y_i, h_{t-1}(x_i) - bh(x_i, a_t))$
5. $H_t(x) = H_{t-1}(x) + b_t h(x, a_t)$
6. **return** H_N

Smoothness of Q

Typical Q is piecewise linear:

$$Q = \sum_{i=1}^{\ell} M = \sum_{i=1}^{\ell} \left[y_i \sum_{t=1}^{N} \alpha_t H_t(x_i) < 0 \right]$$

Smooth approximation of margian loss function $[M \leq 0]$:

- $E(M) = \exp(-M)$ is exponential (in AdaBoost)
- $L(M) = \log_2(1 + e - M)$ is logarithmic (in LogitBoost)
- $Q(M) = (1 - M)^2$ is quadratic (in GentleBoost)
- $G(M) = \exp\big(-cM(M + s)\big)$ is Gaussian (in BrownBoost)

Well-known algorithms

- AdaBoost
- AnyBoost
- LogitBoost
- BrownBoost
- ComBoost
- Stochastic gradient boosting

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

AdaBoost Basis

$$H_T(x) = \sum_{t=1}^{T} b_t h(x, a_t)\,,$$

It is classification, therefore $L(H(x), y) = L(H(x)y)$.

Loss function is $E(M) = \exp(-M)$

Term “weights” appeared earlier than “gradient”.

For weight vector U^ℓ:

- $P(h, U^\ell)$ is the number of correctly classified objects (TP+TN)
- $N(h, U^\ell)$ is the number of incorrectly classified objects (FP+FN)

Main boosting theorem

Theorem (Freund, Schapire, 1995)

For all normalized weights vector U^ℓ, such algorithm $H = h(x, a)$ exist that classifies sample better than randomly:

$$N = N\left(H, U^\ell\right) < 1/2.$$

Then the minimum of $Q^{(t)}$ is reached with

$$H_t = \operatorname{argmin}_H N\left(H, U^\ell\right),$$

$$b_t = \frac{1}{2} \ln \frac{1 - N(H_t, U^\ell)}{N(H_t, U^\ell)}.$$

Objects weights

For $L(H(x), y) = L(H(x)y)$.

$$\nabla Q_i^{(t)} = \frac{\delta L(H_{t-1}(x_i)y_i)}{\delta H_{t-1}(x_i)} = y_i \frac{\delta L(H_{t-1}(x_i)y_i)}{\delta(H_{t-1}(x_i)y_i)} = y_i w_i,$$

where $w_i = \frac{\delta L(H_{t-1}(x_i)y_i)}{\delta(H_{t-1}(x_i)y_i)}$ is a **weight** of object x_i.

Then the forth algorithm step is $a_t =$ Learn$\left(\{x_i\}_{i=1}^{\ell}, \left\{\nabla Q_i^{(t)}\right\}_{i=1}^{\ell}\right)$:

$$h(x, a_t) = \operatorname{argmin}_{a \in A} \sum_{i=1}^{\ell} L\left(h(x_i, a_t), \nabla Q_i^{(t)}\right) =$$

$$= \operatorname{argmin}_{a \in A} \sum_{i=1}^{\ell} L(y_i w_i h(x_i, a_t)).$$

AdaBoost

Input: T^{ℓ}, T

1. **for** $i = 1$ **to** ℓ **do**
2. $\quad w_i = \frac{1}{\ell}$
3. **for** $t = 1$ **to** T **do**
4. $\quad a_t = \mathrm{argmin}_A N\left(h(x, a_t), U^{\ell}\right)$
5. $\quad N_t = \sum_{i=1}^{\ell} w_i [y_i h(x_i . a_t) < 0]$
6. $\quad b_t = \frac{1}{2} \ln \frac{1-N_t}{N_t}$
7. $\quad$ **for** $i = 1$ **to** ℓ **do**
8. $\quad\quad w_i = w_i \exp(-b_t y_t h(x_i, a_t))$
9. $\quad$ NORMALIZE$(\{w_i\}_{i=1}^{\ell})$
10. **return** $H_N = \sum_{t=1}^{T} b_t h(x, a_t)$

Classification refusals

Let $P + N \neq \ell$. The algorithm can **refuse** to classify.

Theorem (Freund, Schapire, 1996)

Let for every normalized weight vector U^ℓ an algorithm $H = h(x, a)$ exists such that it classifies a sample at least a bit better than randomly:

$$N\left(H, U^\ell\right) < P\left(H, U^\ell\right).$$

The minimum of $Q^{(t)}$ is reached with

$$H_t = \operatorname{argmin}_H \sqrt{P(H, U^\ell)} - \sqrt{N(H, U^\ell)},$$

$$b_t = \frac{1}{2} \ln \frac{P(H_t, U^\ell)}{N(H_t, U^\ell)}.$$

Convergence

Theorem (Freund, Schapire, 1996)

If on each step the family H and the learning method allow to synthesize such H_t that

$$\sqrt{P(H, U^\ell)} - \sqrt{N(H, U^\ell)} = \gamma_t > \gamma$$

with a certain $\gamma > 0$, then H_N is built in a fixed number of steps.

What is the number of steps? N, is such that

$$Q^{(1)}(1 - \gamma)^N < 1.$$

Boosting fundamentals

$$\nu_\theta(a, T^\ell) = \frac{1}{\ell}\sum_i^\ell [H(x_i)y_i \le \theta]$$

Theorem (Freund, Schapire, Barlett, 1998)

If $|H| < \infty$, then $\forall\theta > 0$, $\forall\eta \in (0,1)$ with probability $1 - \eta$

$$\Pr[\text{AdaBoost}(x) < 0] \le \nu_\theta(a, T^\ell) + C\sqrt{\frac{\ln|H| \ln \ell}{\ell\theta^2} + \frac{1}{\ell}\ln\frac{1}{\eta}}.$$

It does not depend on T.

Boosting discussion

Advantages:

- hard to get overfitted
- can be applied for different loss functions

Disadvantages:

- no noise processing
- cannot be applied for powerful algorithm
- it is hard to explain result

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Empirical observations

1. Algorithms weights are not very important for achieving equal margins.

2. Objects weights are not very important for achieving difference.

Key idea

Idea: we can build diverse algorithms by learning one model in different conditions.

Precise idea: we can use different bites of dataset.

Synthesis of random algorithms

- **Subsampling:** learn algorithm on subsample.
- **Bagging:** learn algorithm on subsamples of the same length with bootstrap (random choice with returns)
- **Random subspace method:** learn algorithms of subspaces of features
- **Filtering** (next slide)

Filtering

Let we have a sample of infinite size.

Learn first algorithm on X_1, which are first m_1 objects.

Then toss a coin m_2 times:

- head: add in X_2 first incorrectly classified object;
- tail: add in X_2 first correctly classified object.

Learn second algorithm on X_2.

Add in X_3 first m_3 object, on which first two classifiers give different answers.

Learn third algorithm on X_3.

Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Random Forest

For sample $T^\ell = \{x_i, y_i\}_{i=1}^{\ell}$ with n features.

1. Choose a subsample size of ℓ with bootstrap.
2. Synthesize decision tree for that sample, for each vertex choose n' (usually $n' = \sqrt{n}$) random features.
3. No pruning is applied.

This can be done 100, 1000, … times.

Why does it work?

- It is voting
- Trees are easily get overfitted to very different subsamples
- With the growth of the sample, its performance converges

More details

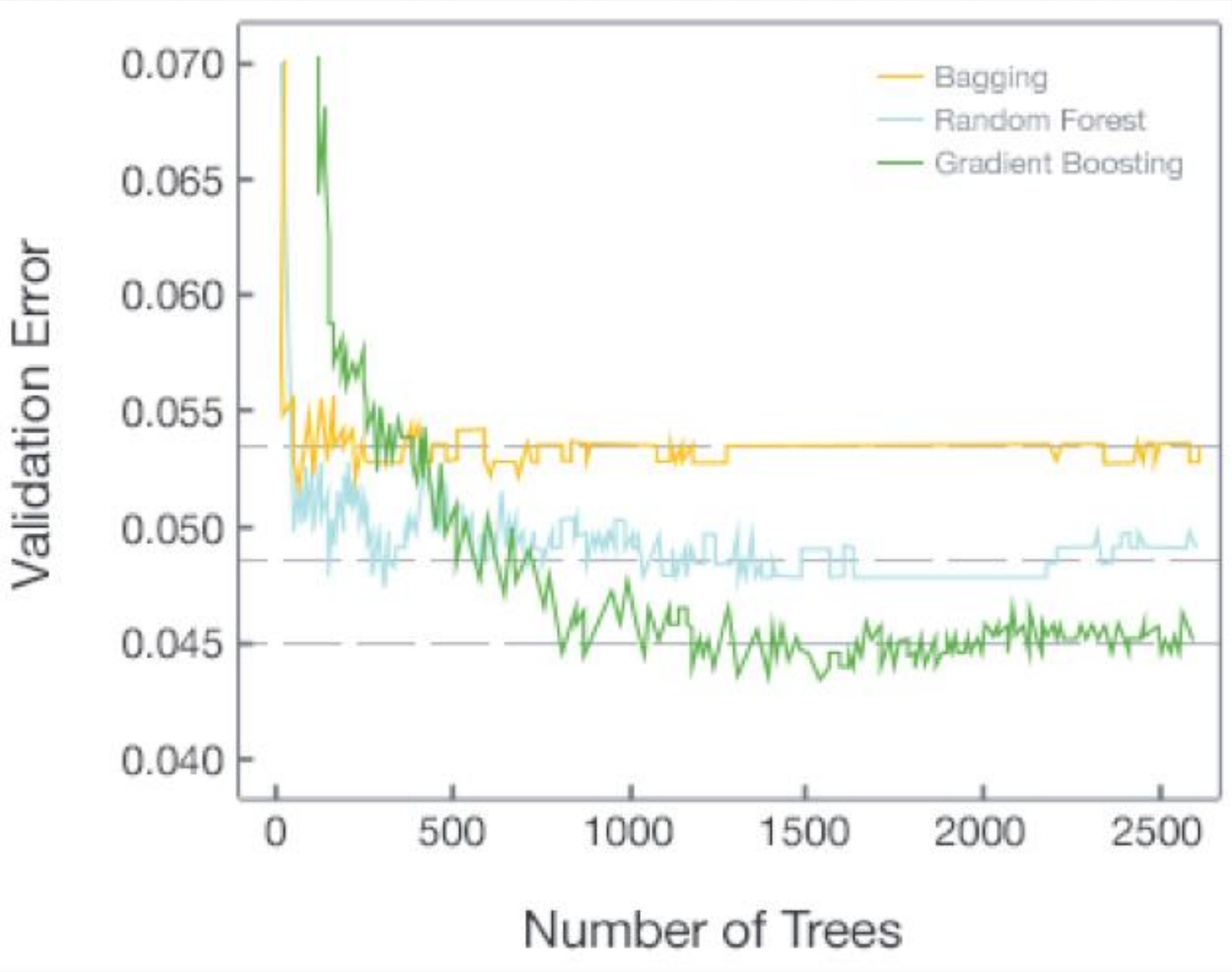
What to combine?

- Simple votes of trees
- Probabilities (frequency of class in the resulting leaf)

How to improve?

- **Extremely randomized trees**: use random values for splitting (it is faster).

Ensemble method comparison



Lecture plan

- Logical rules
- Comprehension
- Rules induction
- Decision trees
- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random Algorithm synthesis
- Random Forest
- Stacking

Stacking key idea

Instead of combining algorithms, use their predictions as new features and learn a model.

This idea can be generalized to using classification results as new features of objects.

Blending key idea

Learn algorithms for stacking on a small (10%) hold-out data subset.

What also can be used?

- Algorithm mixture
- Ranking aggregation
- Model selection
- Combining several ensemble techniques