Lecture 9

Ensemble learning

Machine Learning

Andrey Filchenkov

10.04.2018

Lecture plan

- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random algorithm synthesis
- Stacking

- The presentation is prepared with materials of the K.V. Vorontsov's course "Machine Leaning".
- Slides are available online: **goo.gl/Wkif2w**

Lecture plan

- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random algorithms synthesis
- 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 $p_1, p_2, \dots, p_n \approx p$. 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:

$$P_{vote} = p^n + np^{n-1}(1-p) + \frac{n(n-1)}{2}p^{n-2}(1-p)^2 + \dots +$$
$$+C_n^{n/2}\, p^{n/2}(1-p)^{n/2}.$$

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

- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random algorithms synthesis
- 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

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

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 $Q^{(t)} = \sum_{i=1}^{\ell} L(H_t(x_i), y_i)$ incremently.

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

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

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

We can find it by linear search

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

Generalized algorithm

Input: T^{ℓ}, N

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

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

- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random algorithms synthesis
- 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. $\quad a_t = \mathrm{argmin}_A N\big(h(x, a_t), U^{\ell}\big)$
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. $\quad \textsc{Normalize}(\{w_i\}_{i=1}^{\ell})$
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 foundamentals

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

Theorem (Freund, Schapire, Barlett, 1998)
If $|H| < \infty$, then $\forall\theta > 0$, $\forall\eta \in (0,1)$ with probability $1 - \eta$

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

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

- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random algorithms synthesis
- 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

- Composition of algorithms
- Boosting
- AdaBoost and its theoretical properties
- Random algorithms synthesis
- 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