

Boosting on a Budget: Sampling for Feature-Efficient Prediction

ICML 2011

Lev Reyzin

Georgia Institute of Technology

Motivation

- Looking at **features** may be expensive.
 - medical diagnosis
 - Internet applications
 - etc.
- In many applications, this is especially true during **prediction**, more so than in training.

Model

- Goal is to do supervised learning, using a limited number of features.
 - Given a **budget** on each example, the learner must look at no more features than allowed by the budget.
 - Only **limited in test** data, not training.
 - We call this **feature-efficient** prediction.

Previous/Related Work

- **Sequential analysis:** determining when to stop sequential clinical trials.
 - Wald ('47) and Chernoff ('72)
 - Pelosof et al. ('10) speed up margin-based algorithms
- **PAC learning** analysis with incomplete features.
 - Ben David and Dichterman ('93) and also Greiner et al. ('02)
- Robust prediction with **corrupted/missing features**.
 - Globerson and Roweis ('06)
- Learning **linear hypotheses** without using many features
 - Cesa-Bianchi et al. ('10)
- etc.

Our Idea

The main idea is to modify **boosting** (or any ensemble) for this task by **sampling** from the final ensemble predictor.

A Reminder of Boosting

- Boosting combines many “moderately inaccurate” weak learners into an ensemble predictor.
 - These are often feature-efficient.
- Generates a new **weak learner** on each round.
- Outputs a **weighted distribution** of weak learners (from its hypothesis class).
- Our idea is to **sample** from the distribution over weak learners instead of taking the full vote.
 - If we don’t need too many samples, the final predictor will also be feature efficient.

AdaBoostRS, an Algorithm

- 1) Train AdaBoost [Freund & Schapire '97].
- 2) To **predict** on a new example, instead of taking the full vote, **sample** the weak learners' votes according to their weights.
 - Take as many samples as your budget allows.
- 3) Take the **unweighted** vote of the samples as the prediction.
 - This works best when all features have same cost. We will consider non-uniform costs later in the talk.

Margin Bound

- A **margin** is the weighted fraction of weak learners voting for the correct label.
- [Schapire et al. '98]: for any weighted vote, the generalization error is at most:

the VC dimension
of the base classifier

$$\hat{\Pr} [\text{margin}_f(x, y) \leq \theta] + \tilde{O} \left(\sqrt{\frac{d}{m\theta^2}} \right)$$

Fraction training examples with margin below theta

number of training examples

for any value of theta

(Oversimplified) Intuition On Margins

- If AdaBoost has high margins, then AdaBoostRS doesn't need to sample too many weak learners to be in agreement.
- If AdaBoost has low margins then we don't really care how AdaBoostRS predicts.
- As AdaBoostRS takes more and more samples, in the limit its vote agrees with AdaBoost's.

Some Margin Bounds

- AdaBoost:

$$\mathbf{P}_D[yf(x) \leq 0] \leq \mathbf{P}_S[yf(x) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right)$$

- AdaBoostRS:

$$\mathbf{P}_D[yf(x) \leq 0] \leq \mathbf{P}_S[yf(x) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right) + e^{-N\theta^2/2}$$

d is VC dimension, m is number of training examples, N is number of (weak learner) samples AdaBoostRS takes

More on AdaBoostRS

- Using Decision Stumps as weak learners, each tree/stump looks at only 1 feature. This means we can take as many samples as the budget.
- The situation is even better because of the “birthday paradox.”
 - Some samples are “free.”

Some Experiments

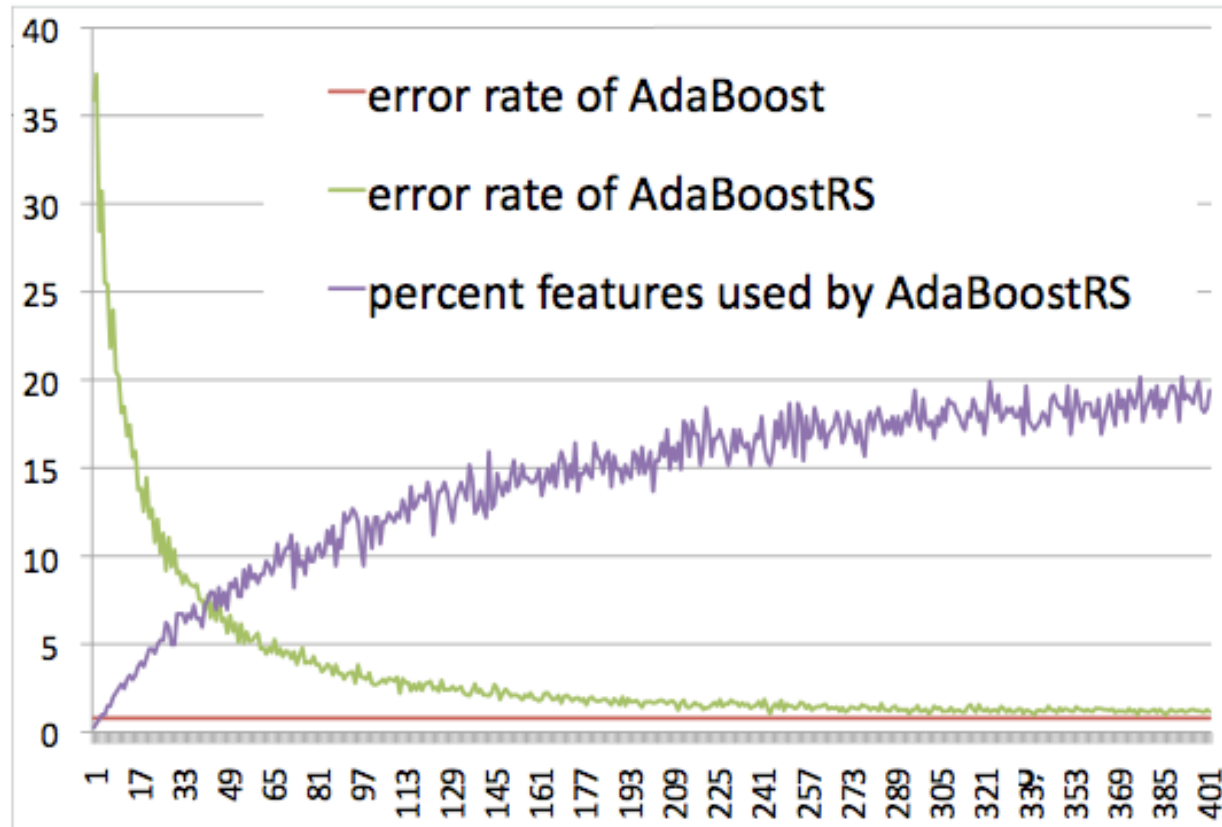


Figure 1. A graph of the error rate of AdaBoostRS on the ocr17 dataset and the percent of features it is using. The horizontal axis is the number of samples drawn by AdaBoostRS.

Some More Experiments

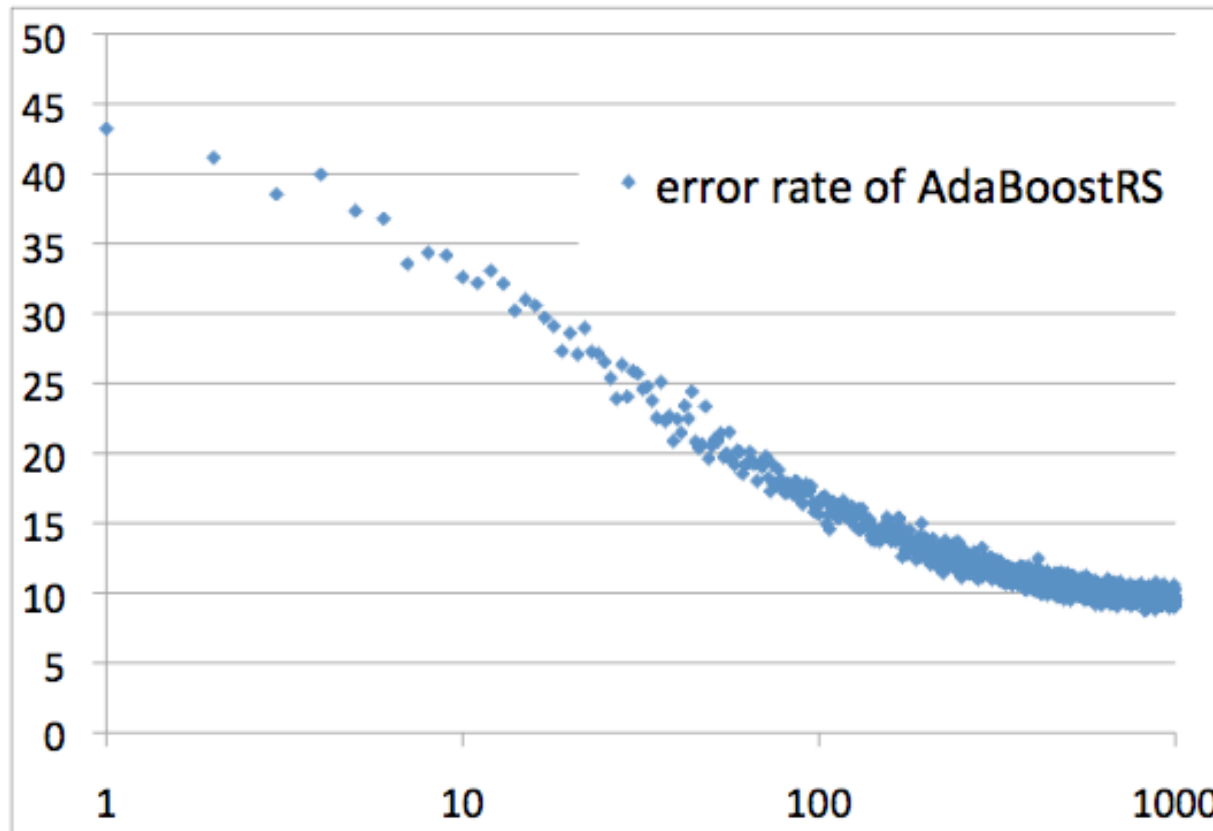


Figure 2. A graph of the error rate of AdaBoostRS on the splice dataset, as a function of the number of samples. The horizontal axis is on a log scale.

Number of Samples Needed to Reach Given Relative Error Rates to AdaBoost.

	100%	50%	25%	10%
census (18.3)	15	77	211	637
splice (8.1)	97	205	421	823
ocr17 (0.8)	178	244	339	505
ocr49 (6.5)	167	296	694	1020

Percent of Features Used to Reach Given Relative Error Rates to AdaBoost.

	100%	50%	25%	10%
census (18.3)	10.0	29.2	40.8	48.5
splice (8.1)	26.8	38.9	52.7	61.5
ocr17 (0.8)	16.4	18.4	19.9	20.6
ocr49 (6.5)	21.6	26.4	31.3	32.3

Non-Uniform Feature Costs

- What if some features are more expensive than others?
- We can modify the sampling procedure to take this into account, using the following probability distribution, making **AdaBoostRS_{AC}**:

$$p(i) = \frac{\alpha_i}{c(h_i) \sum_{i=1}^T (\alpha_t / c(h_t))}$$

- Finally, we can use **importance-weighting** to de-bias the vote.

Error Rates for AdaBoostRS_{AC}

Table 5. Error rates (in percent), averaged over 50 trials, of AdaBoostRS_{AC} and AdaBoostRS using budgets of 10 and 20 when features have random costs drawn i.i.d. from $[0, 1]$. The underlying algorithm, AdaBoost, is run for 500 rounds.

	AdaBoostRS _{AC}	AdaBoostRS
census ($B = 11$)	32.2	32.8
census ($B = 21$)	25.5	26.4
splice ($B = 11$)	25.7	27.0
splice ($B = 21$)	19.2	20.4
ocr17 ($B = 11$)	9.2	10.5
ocr17 ($B = 21$)	3.5	4.3
ocr49 ($B = 11$)	27.4	28.3
ocr49 ($B = 21$)	20.2	21.4

An Explanation for AdaBoostRS_{AC}

Table 6. Number of samples taken (τ), averaged over 50 trials, of AdaBoostRS_{AC} and AdaBoostRS using budgets of 11 and 21 when features have random costs drawn i.i.d. from $[0, 1]$. The underlying algorithm, AdaBoost, is run for 500 rounds.

	AdaBoostRS _{AC}	AdaBoostRS
census ($B = 11$)	26.2	20.7
census ($B = 21$)	45.7	41.3
splice ($B = 11$)	33.8	20.6
splice ($B = 21$)	56.2	40.0
ocr17 ($B = 11$)	29.4	20.6
ocr17 ($B = 21$)	49.3	40.5
ocr49 ($B = 11$)	33.6	21.1
ocr49 ($B = 21$)	55.7	40.3

Discussion

- This method is generic -- AdaBoost can be replaced by your favorite voting algorithm.
- AdaBoostRS's budget does not have to be known in advance.
- Works with a variety of weak learners, can be non-linear.
- Adapts to various feature costs.
- Sampling is “non-adaptive.”

Open Problems

- How does this compare to just taking the “top” weak learners?
- What is the best boosting algorithm to use here?
- Other techniques for feature-efficient prediction?
- Lots of interesting questions left to answer.