AdaBoost

From Wikipedia, the free encyclopedia

AdaBoost, short for "Adaptive Boosting", is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire who won the prestigious "Gödel Prize" in 2003 for their work. It can be used in conjunction with many other types of learning algorithms to improve their performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems, however, it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (i.e., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.

While every learning algorithm will tend to suit some problem types better than others, and will typically have many different parameters and configurations to be adjusted before achieving optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder to classify examples.

Contents

Overview

Problems in machine learning often suffer from the curse of dimensionality — each sample may consist of a huge number of potential features (for instance, there can be 162,336 Haar features, as used by the Viola–Jones object detection framework, in a 24×24 pixel image window), and evaluating every feature can reduce not only the speed of classifier training and execution, but in fact reduce predictive power, per the *Hughes Effect*. Unlike neural

networks and SVMs, the AdaBoost training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features do not need to be computed.

Training

AdaBoost refers to a particular method of training a boosted classifier. A boost classifier is a classifier in the form

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

where each f_t is a weak learner that takes an object x as input and returns a real valued result indicating the class of the object. The sign of the weak learner output identifies the predicted object class and the absolute value gives the confidence in that classification. Similarly, the T-layer classifier will be positive if the sample is believed to be in the positive class and negative otherwise.

Each weak learner produces an output, hypothesis $h(x_i)$, for each sample in the training set. At each iteration t, a weak learner is selected and assigned a coefficient α_t such that the sum training error E_t of the resulting t-stage boost classifier is minimized.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

Here $F_{t-1}(x)$ is the boosted classifier that has been built up to the previous stage of training, $E(F)$ is some error function and $f_t(x) = \alpha_t h(x)$ is the weak learner that is being considered for addition to the final classifier.

Weighting

At each iteration of the training process, a weight is assigned to each sample in the training set equal to the current error $E(F_{t-1}(x_i))$ on that sample. These weights can be used to inform the training of the weak learner, for instance, decision trees can be grown that favor splitting sets of samples with high weights.

Derivation

This derivation follows Rojas (2009):[1]

Suppose we have a data set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ where each item x_i has an associated class $y_i \in \{-1, 1\}$, and a set of weak classifiers $\{k_1, \ldots, k_L\}$ each of which outputs a classification $k_j(x_i) \in \{-1, 1\}$ for each item. After the $m-1$-th iteration our boosted classifier is a linear combination of the weak classifiers of the form:

$$C_{(m-1)}(x_i) = \alpha_1 k_1(x_i) + \cdots + \alpha_{m-1} k_{m-1}(x_i)$$

At the m-th iteration we want to extend this to a better boosted classifier by adding a multiple of one of the weak classifiers:

$$C_m(x_i) = C_{(m-1)}(x_i) + \alpha_m k_m(x_i)$$

So it remains to determine which weak classifier is the best choice for k_m, and what its weight α_m should be. We define the total error E of C_m to be the sum of its exponential loss on each data point, given as follows:

$$E = \sum_{i=1}^{N} e^{-y_i C_m(x_i)}$$

Letting $w_i^{(1)} = 1$ and $w_i^{(m)} = e^{-y_i C_{m-1}(x_i)}$ for $m > 1$, we have:

$$E = \sum_{i=1}^{N} w_i^{(m)} e^{-y_i \alpha_m k_m(x_i)}$$

We can split this summation between those data points that are correctly classified by k_m (so $y_i k_m(x_i) = 1$) and those which are misclassified (so $y_i k_m(x_i) = -1$):

$$E = \sum_{y_i = k_m(x_i)} w_i^{(m)} e^{-\alpha_m} + \sum_{y_i \neq k_m(x_i)} w_i^{(m)} e^{\alpha_m} = \sum_{i=1}^{N} w_i^{(m)} e^{-\alpha_m} + \sum_{y_i \neq k_m(x_i)} w_i^{(m)} (e^{\alpha_m} - e^{-\alpha_m})$$

Since the only part of the right-hand side of this equation that depends on k_m is $\sum_{y_i \neq k_m(x_i)} w_i^{(m)}$, we see that the k_m that minimizes E is the one that minimizes $\sum_{y_i \neq k_m(x_i)} w_i^{(m)}$, i.e. the weak classifier with the lowest weighted error (with weights $w_i^{(m)} = e^{-y_i C_{m-1}(x_i)}$).

In order to determine the desired weight α_m that minimizes E with the k_m that we just determined, we differentiate:

$$\frac{dE}{d\alpha_m} = \sum_{y_i \neq k_m(x_i)} w_i^{(m)} e^{\alpha_m} - \sum_{y_i = k_m(x_i)} w_i^{(m)} e^{-\alpha_m}$$

Setting this to zero and solving for α_m yields:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{\sum_{y_i = k_m(x_i)} w_i^{(m)}}{\sum_{y_i \neq k_m(x_i)} w_i^{(m)}}\right)$$

We calculate the weighted error rate of the weak classifier to be $\epsilon_m = \sum_{y_i \neq k_m(x_i)} w_i^{(m)} / \sum_{i=1}^{N} w_i^{(m)}$, so it follows that:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right)$$

Thus we have derived the AdaBoost algorithm: At each iteration, choose the classifier k_m which minimizes the total weighted error $\sum_{y_i \neq k_m(x_i)} w_i^{(m)}$, use this to calculate the error rate $\epsilon_m = \sum_{y_i \neq k_m(x_i)} w_i^{(m)} / \sum_{i=1}^{N} w_i^{(m)}$, use this to calculate the weight $\alpha_m = \frac{1}{2} \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right)$, and finally use this to improve the boosted classifier C_{m-1} to $C_m = C_{(m-1)} + \alpha_m k_m$.

Statistical understanding of boosting

Boosting is a form of linear regression in which the features of each sample x_i are the outputs of some weak learner h applied to x_i. Specifically, in the case where all weak learners are known a priori, AdaBoost corresponds to a single iteration of the backfitting algorithm in which the smoothing splines are the minimizers of $\sum_{i=1}^{n} e^{-Y_i\hat{\mu}(x_i)} + \infty \int_{x1}^{x_n} \hat{\mu}''(x)^2\,dx$, that is: $\hat{\mu}_i$ fits an exponential cost function and is linear with respect to the observation. Thus, boosting is seen to be a specific type of linear regression.

While regression tries to fit $F(x)$ to $y(x)$ as precisely as possible without loss of generalization, typically using least square error $E(f) = (y(x) - f(x))^2$, the AdaBoost error function $E(f) = e^{-y(x)f(x)}$ takes into account the fact that only the sign of the final result will be used, thus $|F(x)|$ can be far larger than 1 without increasing error. However, the exponential increase in the error for sample x_i as $-y(x_i)f(x_i)$ increases results in excessive weight being assigned to outliers.

One feature of the choice of exponential error function is that the error of the final additive model is the product of the error of each stage, that is, $e^{\sum_i -y_i f(x_i)} = \prod_i e^{-y_i f(x_i)}$. Thus it can be seen that the weight update in the AdaBoost algorithm is equivalent to recalculating the error on $F_t(x)$ after each stage.

There is a lot of flexibility allowed in the choice of loss function. As long as the loss function is monotonic and continuously differentiable, the classifier will always be driven toward purer solutions.[2] Zhang (2004) provides a loss function based on least squares, a modified Huber loss function:

$$\phi(y, f(x)) = \begin{cases} -4yf(x) & \text{if } yf(x) < -1, \\ (yf(x) - 1)^2 & \text{if } -1 \le yf(x) \le 1, \\ 0 & \text{if } yf(x) > 1 \end{cases}$$

This function is more well-behaved than LogitBoost for $f(x)$ close to 1 or -1, does not penalise 'overconfident' predictions ($yf(x) > 1$), unlike unmodified least squares, and only penalises samples misclassified with confidence greater than 1 linearly, as opposed to quadratically or exponentially, and is thus less susceptible to the effects of outliers.

Boosting as Gradient Descent

Boosting can be seen as minimization of a convex loss function over a convex set of functions.[3] Specifically, the loss being minimized by AdaBoost is the exponential loss $\sum_i \phi(i, y, f) = \sum_i e^{-y_i f(x_i)}$, whereas LogitBoost performs logistic regression, minimizing $\sum_i \phi(i, y, f) = \sum_i \ln\left(1 + e^{-y_i f(x_i)}\right)$.

In the gradient descent analogy, the output of the classifier for each training point is considered to be a point $(F_t(x_1), \ldots, F_t(x_n))$ in n-dimensional space, where each axis corresponds to a training sample, each weak learner $h(x)$ corresponds to a vector of fixed orientation and length, and the goal is to reach the target point $(y_1, \ldots, y_n)$ (or any region where the value of loss function $E_T(x_1, \ldots, x_n)$ is less than the value at that

point), in the least number of steps. Thus AdaBoost algorithms perform either Cauchy (find $h(x)$ with the steepest gradient, choose α to minimize test error) or Newton (choose some target point, find $\alpha h(x)$ that will bring F_t closest to that point) optimization of training error.

Example Algorithm (Discrete AdaBoost)

With:

- Samples $x_1 \ldots x_n$
- Desired outputs $y_1 \ldots y_n, y \in \{-1, 1\}$
- Initial weights $w_{1,1} \ldots w_{n,1}$ set to $\frac{1}{n}$
- Error function $E(f(x), y, i) = e^{-y_i f(x_i)}$
- Weak learners $h\colon x \to [-1, 1]$

For t in $1 \ldots T$:

- Choose $f_t(x)$:
  - Find weak learner $h_t(x)$ that minimizes ϵ_t, the weighted sum error for misclassified points $\epsilon_t = \sum_i w_{i,t} E(h_t(x), y, i)$
  - Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
- Add to ensemble:
  - $F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$
- Update weights:
  - $w_{i,t+1} = w_{i,t} e^{-y_i \alpha_t h_t(x_i)}$ for all i
  - Renormalize $w_{i,t+1}$ such that $\sum_i w_{i,t+1} = 1$
  - (Note: It can be shown that $\frac{\sum_{h_{t+1}(x_i)=y_i} w_{i,t+1}}{\sum_{h_{t+1}(x_i)\neq y_i} w_{i,t+1}} = \frac{\sum_{h_t(x_i)=y_i} w_{i,t}}{\sum_{h_t(x_i)\neq y_i} w_{i,t}}$ at every step, which can simplify the calculation of the new weights.)

Choosing α_t

α_t is chosen as it can be analytically shown to be the minimizer of the exponential error function for Discrete AdaBoost.[4]

Minimize:

$$\sum_i w_i e^{-y_i h_i \alpha_t}$$

Using the convexity of the exponential function, and assuming that $\forall i, h_i \in [-1, 1]$ we have:

$$\sum_i w_i e^{-y_i h_i \alpha_t} \leq \sum_i \left(\frac{1-y_i h_i}{2}\right) w_i e^{\alpha_t} + \sum_i \left(\frac{1+y_i h_i}{2}\right) w_i e^{-\alpha_t}$$
$$= \left(\frac{1+\epsilon_t}{2}\right) e^{\alpha_t} + \left(\frac{1-\epsilon_t}{2}\right) e^{-\alpha_t}$$

We then differentiate that expression with respect to α_t and set it to zero to find the minimum of the upper bound:

$$\left(\frac{1+\epsilon_t}{2}\right) e^{\alpha_t} - \left(\frac{1-\epsilon_t}{2}\right) e^{-\alpha_t} = 0$$
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{1+\epsilon_t}\right)$$

Note that this only applies when $h_i \in \{-1, 1\}$, though it can be a good starting guess in other cases, such as when the weak learner is biased ($h(x) \in \{a, b\}, a \neq -b$), has multiple leaves ($h(x) \in \{a, b, \ldots, n\}$) or is some other function $h(x) \in \mathbb{R}$. In such cases the choice of weak learner and coefficient can be condensed to a single step in which $f_t = \alpha_t h_t(x)$ is chosen from all possible α, h as the minimizer of $\sum_i w_{i,t} e^{-y_i f_t(x_i)}$ by some numerical searching routine.

Variants

Real AdaBoost

The output of decision trees is a class probability estimate $p(x) = P(y = 1|x)$, the probability that x is in the positive class.[2] Friedman, Hastie and Tibshirani derive an analytical minimizer for $e^{-y(F_{t-1}(x) + f_t(p(x)))}$ for some fixed $p(x)$ (typically chosen using weighted least squares error):

$$f_t(x) = \frac{1}{2} \ln \left(\frac{x}{1-x}\right).$$

Thus, rather than multiplying the output of the entire tree by some fixed value, each leaf node is changed to output half the logit transform of its previous value.

LogitBoost

LogitBoost represents an application of established logistic regression techniques to the AdaBoost method. Rather than minimizing error with respect to y, weak learners are chosen to minimize the (weighted least-squares) error of $f_t(x)$ with respect to

$$z_t = \frac{y^* - p_t(x)}{2p_t(x)(1-p_t(x))}, \text{ where } p_t(x) = \frac{e^{F_{t-1}(x)}}{e^{F_{t-1}(x)} + e^{-F_{t-1}(x)}}, w_t = p_t(x)(1-p_t(x)) \text{ and}$$
$$y^* = \frac{y+1}{2}.$$

That is z_t is the Newton-Raphson approximation of the minimizer of the log-likelihood error at stage t, and the weak learner f_t is chosen as the learner that best approximates z_t by weighted least squares.

As p approaches either 1 or 0, the value of $p_t(x_i)(1 - p_t(x_i))$ becomes very small and the z term, which will be large for misclassified samples, can become numerically unstable, due to machine precision rounding errors. This can be overcome by enforcing some limit on the absolute value of z and the minimum value of w.

Gentle AdaBoost

While previous boosting algorithms choose f_t greedily, minimizing the overall test error as much as possible at each step GentleBoost features a bounded step size. f_t is chosen to minimize $\sum_i w_{t,i}(y_i - f_t(x_i))^2$, and no further coefficient is applied. Thus, in the case where a weak learner exhibits perfect classification performance, GentleBoost will choose $f_t(x) = \alpha_t h_t(x)$ exactly equal to y, while steepest descent algorithms will try to set $\alpha_t = \infty$. Empirical observations about the good performance of GentleBoost appear to back up Schapire and Singer's remark that allowing excessively large values of α can lead to poor generalization performance.[4][5]

Early Termination

A technique for speeding up processing of boosted classifiers, early termination refers to only testing each potential object with as many layers of the final classifier necessary to meet some confidence threshold, speeding up computation for cases where the class of the object can easily be determined. One such scheme is the object detection framework introduced by Viola and Jones:[6] in an application with significantly more negative samples than positive, a cascade of separate boost classifiers is trained, the output of each stage biased such that some acceptably small fraction of positive samples is mislabeled as negative, and all samples marked as negative after each stage are discarded. If 50% of negative samples are filtered out by each stage, only a very small number of objects would pass through the entire classifier, reducing computation effort. This method has since been generalized, with a formula provided for choosing optimal thresholds at each stage to achieve some desired false positive and false negative rate.[7]

In the field of statistics, where AdaBoost is more commonly applied to problems of moderate dimensionality, early stopping is used as a strategy to reduce overfitting.[8] A validation set of samples is separated from the training set, performance of the classifier on the samples used for training is compared to performance on the validation samples, and training is terminated if performance on the validation sample is seen to decrease even as performance on the training set continues to improve.

Totally Corrective Algorithms

For steepest descent versions of AdaBoost, where α_t is chosen at each layer *t* to minimize test error, the next layer added is said to be *maximally independent* of layer *t*:[9] it is unlikely that a weak learner *t+1* will be chosen that is similar to learner *t*. However, there remains the possibility that *t+1* produces similar information to some other earlier layer. Totally corrective algorithms, such as LPBoost, optimize the value of every coefficient after each step, such that new layers added are always maximally independent of every previous layer. This can be accomplished by backfitting, linear programming or some other method.

Pruning

Pruning refers to the process of removing poorly performing weak classifiers, in order to improve the memory and execution-time cost of the boosted classifier. The simplest methods, which can be particularly effective in conjunction with totally corrective training, are weight- or margin-trimming: when the coefficient, or the contribution to the total test error, of some weak classifier falls below a certain threshold, that classifier is dropped.

Margineantu & Dietterich[10] suggest an alternative criteria for trimming: weak classifiers should be selected such that the diversity of the ensemble is maximized. If two weak learners produce very similar outputs, efficiency can be improved by removing one of them and increasing the coefficient of the remaining weak learner.[11]

See also

- Bootstrap aggregating
- CoBoosting
- BrownBoost
- Gradient boosting

References

1. Rojas, R. (2009). AdaBoost and the super bowl of classifiers a tutorial introduction to adaptive boosting. Freie University, Berlin, Tech. Rep. (http://www.inf.fu-berlin.de/inst/ag-ki/adaboost4.pdf)
2. Friedman, Jerome; Hastie, Trevor; Tibshirani, Robert (1998). "Additive Logistic Regression: A Statistical View of Boosting". CiteSeerX: 10.1.1.51.9525 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.9525).
3. T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization", Annals of Statistics 32 (1), pp. 56-85, 2004.
4. Schapire, Robert; Singer, Yoram (1999). "Improved Boosting Algorithms Using Confidence-rated Predictions". CiteSeerX: 10.1.1.33.4002 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.4002).
5. Freund; Schapire (1999). "A Short Introduction to Boosting" (http://www.site.uottawa.ca/~stan/csi5387/boost-tut-ppr.pdf) (PDF):
6. Viola, Paul; Jones, Robert (2001). "Rapid Object Detection Using a Boosted Cascade of Simple Features". CiteSeerX: 10 .1.1.10.6807 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.6807).
7. McCane, Brendan; Novins, Kevin; Albert, Michael (2005). "Optimizing cascade classifiers.".
8. Trevor Hastie, Robert Tibshirani, Jerome (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (http://statweb.stanford.edu/~tibs/ElemStatLearn/download.html) (2nd ed.). New York: Springer. ISBN 978-0-387-84858-7.
9. Šochman, Jan; Matas, Jiří (2004). "Adaboost with Totally Corrective Updates for Fast Face Detection". ISBN 0-7695-2122-3.
10. Margineantu, Dragos; Dietterich, Thomas (1997). "Pruning Adaptive Boosting". CiteSeerX: 10.1.1.38.7017 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.7017).
11. Tamon, Christino; Xiang, Jie (2000). "On the Boosting Pruning Problem".

Implementations

- AdaBoost and the Super Bowl of Classifiers - A Tutorial on AdaBoost. (http://www.inf.fu-berlin.de/inst/ag-ki/adaboost4.pdf)
- AdaBoost in C++ (http://codingplayground.blogspot.com/2009/03/adaboost-improve-your-performance.html), an implementation of AdaBoost in C++ and boost by Antonio Gulli
- icsiboost (http://code.google.com/p/icsiboost/), an open source implementation of Boostexter
- JBoost (http://jboost.sourceforge.net), a site offering a classification and visualization package, implementing AdaBoost among other boosting algorithms.
- MATLAB AdaBoost toolbox. Includes Real AdaBoost, Gentle AdaBoost and Modest AdaBoost implementations. (http://graphics.cs.msu.ru/en/science/research/machinelearning/AdaBoosttoolbox)
- A Matlab Implementation of AdaBoost (http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=21317&objectType=file)
- Multi-threaded MATLAB-compatible implementation of Boosted Trees (https://sites.google.com/site/carlosbecker/resources/gradient-boosting-boosted-trees)
- milk (http://luispedro.org/software/milk) for Python implements AdaBoost (http://packages.python.org/milk/AdaBoost.html).
- MPBoost++ (http://www.esuli.it/mpboost), a C++ implementation of the original AdaBoost.MH algorithm

and of an improved variant, the MPBoost algorithm.
- multiboost (http://www.multiboost.org/), a fast C++ implementation of multi-class/multi-label/multi-task boosting algorithms. It is based on AdaBoost.MH but also implements popular cascade classifiers and FilterBoost along with a batch of common multi-class base learners (stumps, trees, products, Haar filters).
- NPatternRecognizer (http://npatternrecognizer.codeplex.com/) , a fast machine learning algorithm library written in C#. It contains support vector machine, neural networks, bayes, boost, k-nearest neighbor, decision tree, ..., etc.
- OpenCV implementation of several boosting variants (http://docs.opencv.org/modules/ml/doc/boosting.html)
- Into (https://github.com/topiolli/into) contains open source implementations of many AdaBoost and FloatBoost variants in C++.
- Mallet (http://mallet.cs.umass.edu/) Java implementation.
- adabag (http://cran.r-project.org/web/packages/adabag/) adabag: An R package for binary and multiclass Boosting and Bagging.
- Scikit-learn (http://scikit-learn.org/dev/modules/ensemble.html#AdaBoost) Python implementation.
- fertilized forests (http://www.fertilized-forests.org) A general purpose, platform independent, easy to extend decision forest library that supports boosted training based on multiclass AdaBoost.M2, Samme and Samme.R

External links

- "A decision-theoretic generalization of on-line learning and an application to boosting". *Journal of Computer and System Sciences* **55**. 1997. doi:10.1006/jcss.1997.1504 (https://dx.doi.org/10.1006%2Fjcss.1997.1504). CiteSeerX: 10.1.1.32.8918 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.8918): original paper of Yoav Freund and Robert E.Schapire where AdaBoost is first introduced.
- "Boosting.org" (http://www.boosting.org): a site on boosting and related ensemble learning methods
- "AdaBoost" (http://cmp.felk.cvut.cz/~sochmj1/adaboost_talk.pdf) (PDF): presentation summarizing AdaBoost (see page 4 for an illustrated example of performance).
- "AdaBoost example" (https://www.flll.jku.at/sites/default/files/Adaboost.odp): presentation showing an AdaBoost example.
- Freund; Schapire (1999). "A Short Introduction to Boosting" (http://www.site.uottawa.ca/~stan/csi5387/boost-tut-ppr.pdf) (PDF): introduction to AdaBoost
- "An applet demonstrating AdaBoost" (http://www.cs.ucsd.edu/~yfreund/AdaBoost/index.html).
- Polikar, R. (2006). "Ensemble Based Systems in Decision Making" (http://engineering.rowan.edu/~polikar/RESEARCH/PUBLICATIONS/csm06.pdf) (PDF). *IEEE Circuits and Systems Magazine* **6** (3): 21–45: a tutorial article on ensemble systems including pseudocode, block diagrams and implementation issues for AdaBoost and other ensemble learning algorithms.