Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Add links

# LPBoost

From Wikipedia, the free encyclopedia

**Linear Programming Boosting** (**LPBoost**) is a supervised classifier from the boosting family of classifiers. LPBoost maximizes a *margin* between training samples of different classes and hence also belongs to the class of margin-maximizing supervised classification algorithms. Consider a classification function

$$f : \mathcal{X} \to \{-1, 1\},$$

which classifies samples from a space $\mathcal{X}$ into one of two classes, labelled 1 and -1, respectively. LPBoost is an algorithm to *learn* such a classification function given a set of training examples with known class labels. LPBoost is a machine learning technique and especially suited for applications of joint classification and feature selection in structured domains.



## LPBoost overview [edit]

As in all boosting classifiers, the final classification function is of the form

$$f(\boldsymbol{x}) = \sum_{j=1}^{J} \alpha_j h_j(\boldsymbol{x}),$$

where $\alpha_j$ are non-negative weightings for *weak* classifiers $h_j : \mathcal{X} \to \{-1, 1\}$. Each individual weak classifier $h_j$ may be just a little bit better than random, but the resulting linear combination of many weak classifiers can perform very well.

LPBoost constructs $f$ by starting with an empty set of weak classifiers. Iteratively, a single weak classifier to add to the set of considered weak classifiers is selected, added and all the weights $\boldsymbol{\alpha}$ for the current set of weak classifiers are adjusted. This is repeated until no weak classifiers to add remain.

The property that all classifier weights are adjusted in each iteration is known as *totally-corrective* property. Early boosting methods, such as AdaBoost do not have this property and converge slower.

## Linear program [edit]

More generally, let $\mathcal{H} = \{h(\cdot;\omega) | \omega \in \Omega\}$ be the possibly infinite set of weak classifiers, also termed *hypotheses*. One way to write down the problem LPBoost solves is as a linear program with infinitely many variables.

The primal linear program of LPBoost, optimizing over the non-negative weight vector $\boldsymbol{\alpha}$, the non-negative vector $\boldsymbol{\xi}$ of slack variables and the *margin* $\rho$ is the following.

$$\begin{aligned}
\min_{\boldsymbol{\alpha},\boldsymbol{\xi},\rho} \quad & -\rho + D\sum_{n=1}^{\ell}\xi_n \\
\text{sb.t.} \quad & \textstyle\sum_{\omega\in\Omega} y_n\alpha_\omega h(\boldsymbol{x}_n;\omega) + \xi_n \geq \rho, \qquad n = 1,\ldots,\ell, \\
& \textstyle\sum_{\omega\in\Omega}\alpha_\omega = 1, \\
& \xi_n \geq 0, \qquad n = 1,\ldots,\ell, \\
& \alpha_\omega \geq 0, \qquad \omega\in\Omega, \\
& \rho\in\mathbb{R}.
\end{aligned}$$

Note the effects of slack variables $\boldsymbol{\xi}\geq 0$: their one-norm is penalized in the objective function by a constant factor $D$, which—if small enough—always leads to a primal feasible linear program.

Here we adopted the notation of a parameter space $\Omega$, such that for a choice $\omega\in\Omega$ the weak classifier $h(\cdot;\omega):\mathcal{X}\to\{-1,1\}$ is uniquely defined.

When the above linear program was first written down in early publications about boosting methods it was disregarded as intractable due to the large number of variables $\boldsymbol{\alpha}$. Only later it was discovered that such linear programs can indeed be solved efficiently using the classic technique of column generation.

## Column Generation for LPBoost [edit]

In a linear program a *column* corresponds to a primal variable. Column generation is a technique to solve large linear programs. It typically works in a restricted problem, dealing only with a subset of variables. By generating primal variables iteratively and on-demand, eventually the original unrestricted problem with all variables is recovered. By cleverly choosing the columns to generate the problem can be solved such that while still guaranteeing the obtained solution to be optimal for the original full problem, only a small fraction of columns has to be created.

### LPBoost dual problem [edit]

Columns in the primal linear program corresponds to rows in the dual linear program. The equivalent dual linear program of LPBoost is the following linear program.

$$\begin{aligned}
\max_{\boldsymbol{\lambda},\gamma} \quad & \gamma \\
\text{sb.t.} \quad & \textstyle\sum_{n=1}^{\ell} y_n h(\boldsymbol{x}_n;\omega)\lambda_n + \gamma \leq 0, \qquad \omega\in\Omega, \\
& 0 \leq \lambda_n \leq D, \qquad n = 1,\ldots,\ell, \\
& \textstyle\sum_{n=1}^{\ell}\lambda_n = 1, \\
& \gamma\in\mathbb{R}.
\end{aligned}$$

For linear programs the optimal value of the primal and dual problem are equal. For the above primal and dual problems, the optimal value is equal to the negative 'soft margin'. The soft margin is the size of the margin separating positive from negative training instances minus positive slack variables that carry penalties for margin-violating samples. Thus, the soft margin may be positive although not all samples are linearly separated by the classification function. The latter is called the 'hard margin' or 'realized margin'.

### Convergence criterion [edit]

Consider a subset of the satisfied constraints in the dual problem. For any finite subset we can solve the linear program and thus satisfy all constraints. If we could prove that of all the constraints which we did not add to the dual problem no single constraint is violated, we would have proven that solving our restricted problem is equivalent to solving the original problem. More formally, let $\gamma^*$ be the optimal objective function value for any restricted instance. Then, we can formulate a search problem for the 'most violated constraint' in the original problem space, namely finding $\omega^*\in\Omega$ as

$$\omega^* = \operatorname*{argmax}_{\omega\in\Omega}\sum_{n=1}^{\ell} y_n h(\boldsymbol{x}_n;\omega)\lambda_n.$$

That is, we search the space $\mathcal{H}$ for a single decision stump $h(\cdot;\omega^*)$ maximizing the left hand side of the dual constraint. If the constraint cannot be violated by any choice of decision stump, none of the corresponding constraint can be active in the original problem and the restricted problem is equivalent.

### Penalization constant $D$ [edit]

The positive value of penalization constant $D$ has to be found using model selection techniques. However, if we choose $D = \frac{1}{\ell\nu}$, where $\ell$ is the number of training samples and $0<\nu<1$, then the new parameter $\nu$ has the following properties.

- $\nu$ is an upper bound on the fraction of training errors; that is, if $k$ denotes the number of misclassified training samples, then $\frac{k}{\ell} \leq \nu$.
- $\nu$ is a lower bound on the fraction of training samples outside or on the margin.

## Algorithm [edit]

- Input:
  - Training set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}, \boldsymbol{x}_i \in \mathcal{X}$
  - Training labels $Y = \{y_1, \ldots, y_\ell\}, y_i \in \{-1, 1\}$
  - Convergence threshold $\theta \geq 0$
- Output:
  - Classification function $f : \mathcal{X} \to \{-1, 1\}$

1. Initialization
   1. Weights, uniform $\lambda_n \leftarrow \frac{1}{\ell}, \quad n = 1, \ldots, \ell$
   2. Edge $\gamma \leftarrow 0$
   3. Hypothesis count $J \leftarrow 1$
2. Iterate
   1. $\hat{h} \leftarrow \underset{\omega \in \Omega}{\operatorname{argmax}} \sum_{n=1}^{\ell} y_n h(\boldsymbol{x}_n; \omega) \lambda_n$
   2. if $\sum_{n=1}^{\ell} y_n \hat{h}(\boldsymbol{x}_n) \lambda_n + \gamma \leq \theta$ then
      1. break
   3. $h_J \leftarrow \hat{h}$
   4. $J \leftarrow J + 1$
   5. $(\boldsymbol{\lambda}, \gamma) \leftarrow$ solution of the LPBoost dual
   6. $\boldsymbol{\alpha} \leftarrow$ Lagrangian multipliers of solution to LPBoost dual problem
3. $f(\boldsymbol{x}) := \operatorname{sign}\left(\sum_{j=1}^{J} \alpha_j h_j(\boldsymbol{x})\right)$

Note that if the convergence threshold is set to $\theta = 0$ the solution obtained is the global optimal solution of the above linear program. In practice, $\theta$ is set to a small positive value in order obtain a good solution quickly.

### Realized margin [edit]

The actual margin separating the training samples is termed the *realized margin* and is defined as

$$\rho(\boldsymbol{\alpha}) := \min_{n=1,\ldots,\ell} y_n \sum_{\alpha_\omega \in \Omega} \alpha_\omega h(\boldsymbol{x}_n; \omega).$$

The realized margin can and will usually be negative in the first iterations. For a hypothesis space that permits singling out of any single sample, as is commonly the case, the realized margin will eventually converge to some positive value.

### Convergence guarantee [edit]

While the above algorithm is proven to converge, in contrast to other boosting formulations, such as AdaBoost and TotalBoost, there are no known convergence bounds for LPBoost. In practise however, LPBoost is known to converge quickly, often faster than other formulations.

## Base learners [edit]

LPBoost is an ensemble learning method and thus does not dictate the choice of base learners, the space of hypotheses $\mathcal{H}$. Demiriz et al. showed that under mild assumptions, any base learner can be used. If the base learners are particularly simple, they are often referred to as *decision stumps*.

The number of base learners commonly used with Boosting in the literature is large. For example, if $\mathcal{X} \subseteq \mathbb{R}^n$, a base learner could be a linear soft margin support vector machine. Or even more simple, a simple stump of the form

$$h(\boldsymbol{x}; \omega \in \{1,-1\}, p \in \{1,\dots,n\}, t \in \mathbb{R}) := \begin{cases} \omega & \text{if } \boldsymbol{x}_p \leq t \\ -\omega & \text{otherwise} \end{cases}.$$

The above decision stumps looks only along a single dimension $p$ of the input space and simply thresholds the respective column of the sample using a constant threshold $t$. Then, it can decide in either direction, depending on $\omega$ for a positive or negative class.

Given weights for the training samples, constructing the optimal decision stump of the above form simply involves searching along all sample columns and determining $p$, $t$ and $\omega$ in order to optimize the gain function.

## References [edit]


- Linear Programming Boosting via Column Generation, A. Demiriz and K.P. Bennett and J. Shawe-Taylor. Published 2002 in Kluwer Machine Learning 46, pages 225–254.


Categories: Ensemble learning