Adaptive basis function models

Jiali Lin

Virginia Tech

December 4, 2016

Outline

Introduction

Classification and regression trees (CART)

Boosting

Ensemble learning

Introduction

- Consider kernel methods to create non-linear models for regression and classification.
- The prediction takes the form $f(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x})$, where $\phi(\boldsymbol{x})$ is kernel.
- Coming up with a good kernel function is quite difficult.
- **Goal:** dispense with kernels altogether, and try to learn useful features $\phi(\boldsymbol{x})$ directly from the input data.
- **Adaptive basis function model (ABM)** has the form

$$f(\boldsymbol{x}) = w_0 + \sum_{m=1}^{M} w_m \phi_m(\boldsymbol{x})$$

Outline

Introduction

Classification and regression trees (CART)

Boosting

Ensemble learning

Basics

Consider the tree in Figure.

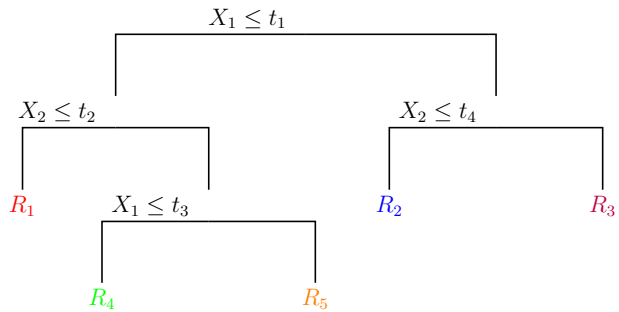


Figure: The first node asks if x_1 is less than some threshold t_1. If yes, we then ask if x2 is less than some other threshold t_2. If yes, we are in the bottom left quadrant of space, R_1. If no, we ask if x_1 is less than t_3. And so on. The result of these axis parallel splits is to partition 2d space into 5 regions.

We can write the model in the following form

$$f(\boldsymbol{x}) = E[y|\boldsymbol{x}] = \sum_{m=1}^{M} w_m \mathbb{I}(\boldsymbol{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\boldsymbol{x}; \boldsymbol{v}_m)$$

where R_m is the m'th region, w_m is the mean response in this region, and $\boldsymbol{v}_m$ encodes the choice of variable to split on.

Growing a tree

- Finding the optimal partitioning of the data is NP-complete.
- Methods to compute a locally optimal MLE: **CART**, **C4.5** and **ID3**.
- The split function chooses the best feature, and the best value for that feature

$$(j^*, t^*) = \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \text{cost}(\boldsymbol{x}_i, y_i : x_{ij} \leq t) + \text{cost}(\boldsymbol{x}_i, y_i : x_{ij} > t)$$

  where $\mathcal{T}_j$ is set of possible thresholds for feature j.
- Split function checks if a node is worth splitting can use several stopping heuristics.
- Cost measure depends on whether our goal is regression or classification.

Cost

- **Regression cost**

$$\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2$$

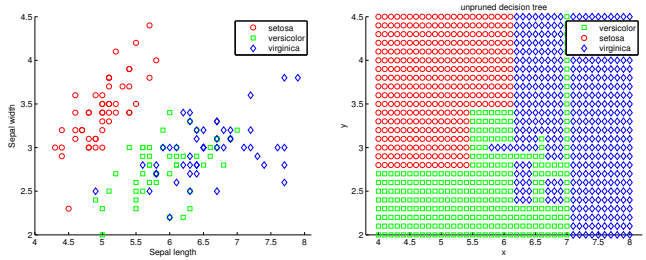
  where $\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i$.

- **Classification cost:** First, we fit a multinoulli model to the data in the leaf satisfying the test $X_j < t$ by estimating

$$\hat{\pi_c} = \frac{1}{|D|} \sum_{i \in D} \mathbb{I}(y_i = c)$$

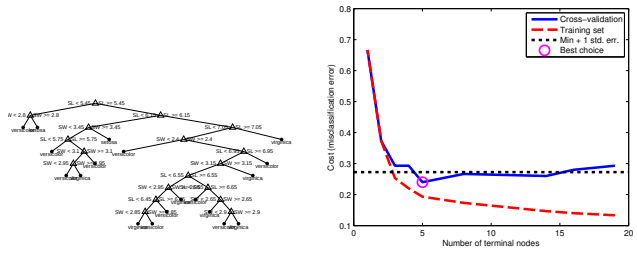
  Several ways to measure the quality of a split:
  - **M**isclassification rate.
  - Entropy, or deviance.
  - Information gain.
  - Gini index.

Example



(a) (b)

Figure: (a) Iris data. We only show the first two features, sepal length and sepal width, and ignore petal length and petal width. (b) Decision boundaries induced by the decision tree in Figure 2(a).



(a) (b)

Figure: (a) Unpruned decision tree for Iris data. (b) Plot of misclassification error rate vs depth of tree. Figure generated by `dtreeDemoIris`.

Pruning a tree

- Motivation: prevent overfitting.
- Idea: grow a "full" tree, and then to perform **pruning**.
- How far to prune back? Evaluate the cross-validated error on each subtree.
- Biggest **cons**: unstable! outlier can leads to high variance trees.

Random forests

- Reduce the variance of an estimate: average together many estimates (say M trees)

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \frac{1}{M} f_m(\boldsymbol{x})$$

  where f_m is the m'th tree (This is called **bagging**).
- Problems: can result in highly correlated predictors.
- **Random forests** decorrelates the base learners by learning trees based on a randomly chosen subset of input variables and datasets.

Outline

Introduction

Classification and regression trees (CART)

Boosting

Ensemble learning

Introduction

- **Boosting:** a greedy algorithm for fitting ABMs, where the ϕ_m are generated by an algorithm called a **weak learner** or a **base learner**.
- Idea: applying the weak learner sequentially to weighted versions of the data.
- More weight is given to examples that were misclassified by earlier rounds.
- **Goal:** solve the optimization problem

$$\min_f \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i))$$

Name	Loss	Derivative	Minimizers f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\boldsymbol{x}_i))^2$	$y_i - f(\boldsymbol{x}_i)$	$E[Y\|\boldsymbol{x}_i]$	L2Boosting
Absolute error	$\|y_i - f(\boldsymbol{x}_i)\|$	sign$(y_i - f(\boldsymbol{x}_i))$	median$(y\|\boldsymbol{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\boldsymbol{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\boldsymbol{x}_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

Table: Some commonly used loss functions. Assume $\tilde{y}_i \in \{-1, +1\}$, $y_i \in \{0, 1\}$ and $\pi_i = \text{sigm}(2f(\boldsymbol{x}_i))$. For regression problems, we assume $y_i \in \mathbb{R}$.

Generalized additive models

- Finding the optimal f may be hard, we shall tackle it sequentially.
- We initialise by defining

$$f_0(\boldsymbol{x}) = \min_{\boldsymbol{\gamma}} \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i; \boldsymbol{\gamma}))$$

- Then at iteration m, we compute

$$(\beta_m, \boldsymbol{\gamma}_m) = \min_{\beta, \boldsymbol{\gamma}} \sum_{i=1}^{N} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta\phi(\boldsymbol{x}_i; \boldsymbol{\gamma}))$$

- Then, we set

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m\phi(\boldsymbol{x}; \boldsymbol{\gamma}_m)$$

- **Key:** do not go back and adjust earlier parameters. This is called **forward stagewise additive modeling**.
- In practice, we do "partial updates"

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu\beta_m\phi(\boldsymbol{x}; \boldsymbol{\gamma}_m)$$

where $0 < \nu \leq 1$ is a step-size parameter (called **shrinkage**).

L2boosting

- Suppose we used squared error loss. Then at step m the loss

$$L(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta\phi(\boldsymbol{x}_i; \boldsymbol{\gamma})) = (r_{im} - \phi(\boldsymbol{x}_i; \boldsymbol{\gamma}))^2$$

  where $r_{im} = y_i - f_{m-1}(\boldsymbol{x}_i)$ is the current residual, and we have set $\beta = 1$.
- We can find the new basis function by using the weak learner to predict $\boldsymbol{r}_m$.

AdaBoost

- Consider a binary classification problem with exponential loss. At step m we have to minimize

$$L_m(\phi) = \sum_{i=1}^{N} \exp[-\tilde{y}_i(f_{m-1}(\boldsymbol{x}) + \beta\phi(\boldsymbol{x}_i))] = \sum_{i=1}^{N} w_{i,m} \exp(-\beta\tilde{y}_i\phi(\boldsymbol{x}_i))$$

  where $w_{i,m} = \exp(-\tilde{y}_i(f_{m-1}(\boldsymbol{x}))$ is a weight applied to datacase i.

- Rewrite this objective as follows

$$\begin{aligned} L_m &= e^{-\beta} \sum_{\tilde{y}_i = \phi(\boldsymbol{x}_i)} w_{i,m} + e^{\beta} \sum_{\tilde{y}_i \neq \phi(\boldsymbol{x}_i)} w_{i,m} \\ &= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} \mathbb{I}(\tilde{y}_i \neq \phi(\boldsymbol{x}_i)) + e^{-\beta} \sum_{i=1}^{N} w_{i,m} \end{aligned}$$

AdaBoost (cont'd)

- Consequently the optimal function to add is

$$\phi_m = \operatorname*{argmin}_{\phi} w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi(\boldsymbol{x}_i))$$

- After finding ϕ_m, subsitute ϕ_m into L_m

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathsf{err}_m}{\mathsf{err}_m}$$

  where

$$\mathsf{err}_m = \frac{\sum_{i=1}^{N} w_i (\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i))}{\sum_{i=1}^{N} w_{i,m}}$$

- The overall updates becomes

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m \phi_m(\boldsymbol{x})$$

AdaBoost (cont'd)

The weights at the next iteration become

$$\begin{aligned} w_{i,m+1} &= w_{i,m} e^{-\beta_m \tilde{y}_i \phi_m(\boldsymbol{x}_i)} \\ &= w_{i,m} e^{\beta_m (2\mathbb{I}(\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i)) - 1)} \\ &= w_{i,m} e^{2\beta_m 2\mathbb{I}(\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i))} e^{-\beta_m} \end{aligned}$$

where we exploited that $-\tilde{y}_i \phi_m(\boldsymbol{x}_i) = -1$ if $\tilde{y}_i = \phi_m(\boldsymbol{x}_i)$ and $-\tilde{y}_i \phi_m(\boldsymbol{x}_i) = +1$ otherwise. Since $e^{-\beta_m}$ will cancel out in the normalization step, we can drop it.

---

given $w_i = 1/N$.

for $m = 1 : M$ **do**

1. Fit a classifier $\phi_m(\boldsymbol{x})$ to the training set using weights w.
2. Compute $\text{err}_m = \frac{\sum_{i=1}^N w_i I(\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i))}{\sum_{i=1}^N w_{i,m}}$.
3. Compute $\alpha_m = \log \frac{1-\text{err}_m}{\text{err}_m}$.
4. Set $w_i \leftarrow w_i \exp\left[\alpha_m \mathbb{I}(\tilde{y}_i \neq \phi_m(\boldsymbol{x}_i))\right]$.

return $f(\boldsymbol{x}) = \text{sgn}\left[\sum_{m=1}^M \alpha_m \phi_m(\boldsymbol{x}))\right]$.

Why does boosting work so well?

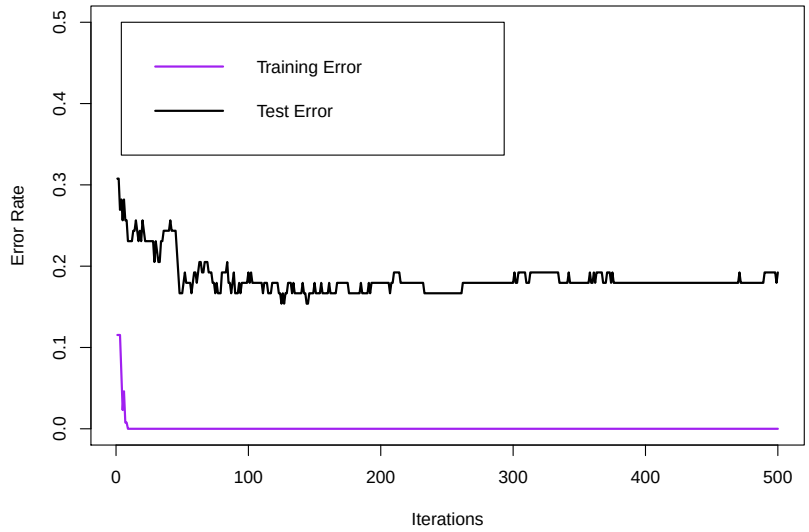


Figure: Illustration of AdaBoost. AdaBoost is very slow to overfit. Figure generated by `AdaboostDemo`.

- Can be viewed as "ℓ_1 regularization": use ℓ_1 regularization to select a subset, $\phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \ldots, \phi_K(\boldsymbol{x})]$.
- It can be proved that AdaBoost maximizes the margin on the training data.

LogitBoost

- Exponential loss puts a lot of weight on misclassified examples.
- Therefore, exponential loss is sensitive to outliers (mislabeled examples).
- Cannot recover probability estimates from $f(\boldsymbol{x})$, $\exp(-\tilde{y}_i f(\boldsymbol{x}_i))$ ss not the logarithm of any pmf for binary variables $\tilde{y}_i \in \{-1, +1\}$.

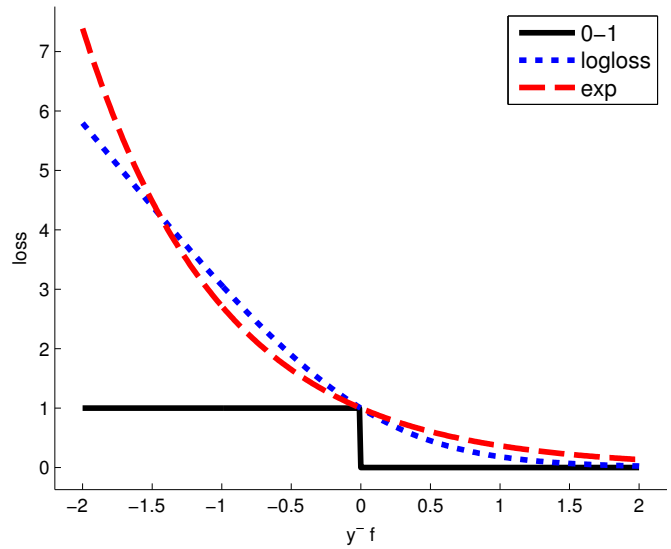


Figure: Illustration of various loss functions for binary classification. Figure generated by `hingeLossPlot`.

LogitBoost (cont'd)

- **Logloss** only punishes mistakes linearly.
- It means we will be able to extract probabilities from the final learned function, by

$$p(y=1|\boldsymbol{x}) = \frac{e^{f(\boldsymbol{x})}}{e^{-f(\boldsymbol{x})} + e^{f(\boldsymbol{x})}} = \frac{1}{e^{-2f(\boldsymbol{x})} + 1}$$

- The goal is to minimze the expected log-loss, given by

$$L_m(\phi) = \sum_{i=1}^{N} \log\left[1 + \exp(-2\tilde{y}_i(f_{m-1}(\boldsymbol{x}) + \phi(\boldsymbol{x}_i)))\right]$$

---

given $w_i = 1/N, \pi_i = 1/2$.

for $m = 1 : M$ **do**

1. Compute the working response $z_i = \frac{y^* - \pi}{\pi_i(1-\pi_i)}$.
2. Compute the weights $w_i = \pi(1-\pi)$.
3. Compute $\phi_m = \mathrm{argmin}_\phi \sum_{i=1}^{N} w_i (z_i - \phi(\boldsymbol{x}_i))^2$.
4. Update $f(\boldsymbol{x}) \leftarrow f(\boldsymbol{x}) + \frac{1}{2}\phi_m(\boldsymbol{x})$.
5. Compute $\pi = 1/(e^{-2f(\boldsymbol{x}_i)} + 1)$.

return $f(\boldsymbol{x}) = \mathrm{sgn}\left[\sum_{m=1}^{M} \phi_m(\boldsymbol{x}))\right]$.

Boosting as functional gradient descent

- **Gradient boosting**: imagine minimizing

$$\hat{\boldsymbol{f}} = \min_{\boldsymbol{f}} L(\boldsymbol{f})$$

  where $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N))$ are the "parameters".
- At step m let $\boldsymbol{g}_m$ be the gradient of $L(\boldsymbol{f})$ evaluated at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$:

$$g_{im} = [\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}]_{f=f_{m-1}}$$

- Then make the update

$$\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m$$

  where ρ_m is the step length, chosen by

$$\rho_m = \min_{\rho} L(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m)$$

Boosting as functional gradient descent (cont'd)

- We only optimizes f, but do not learn a function that can generalize.
- Modify the algorithm by fitting a weak learner to approximate the negative gradient signal

$$\boldsymbol{\gamma}_m = \min_{\boldsymbol{\gamma}} \sum_{i=1}^{N} (-g_{im} - \phi(\boldsymbol{x}_i, \boldsymbol{\gamma}))^2$$

Initialize $f_0(\boldsymbol{x}) = \min_{\boldsymbol{\gamma}} \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i))$.

for $m = 1 : M$ **do**

1. Compute the gradient residual using $r_{im} = -[\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}]_{f(\boldsymbol{x}_i)=f_{m-1}(\boldsymbol{x}_i)}$.
2. Compute the weights $w_i = \pi(1-\pi)$.
3. Use the weak learner to compute $\boldsymbol{\gamma}_m$ which minimizes $\sum_{i=1}^{N}(r_{im} - \phi(\boldsymbol{x}_i, \boldsymbol{\gamma}))^2$.
4. Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu\phi(\boldsymbol{x}; \boldsymbol{\gamma}_m)$.

return $f(\boldsymbol{x}) = f_M(\boldsymbol{x})$.

Outline

Introduction

Classification and regression trees (CART)

Boosting

Ensemble learning

Stacking

- **Ensemble learning:** learning a weighted combination of base models

$$f(y|\boldsymbol{x}, \boldsymbol{\pi}) = \sum_{m \in M} w_m f_m(y|\boldsymbol{x})$$

- Estimate the weights

$$\hat{\boldsymbol{w}} = \min_{\boldsymbol{w}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} w_m f_m(\boldsymbol{x}))$$

- However, this will result in overfitting, with w_m being large for the most complex model.
- **Solution:** use the LOOCV estimate

$$\hat{\boldsymbol{w}} = \min_{\boldsymbol{w}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} w_m f_m^{-1}(\boldsymbol{x})).$$

  where $f_m^{-1}(\boldsymbol{x})$ is the predictor obtained by training on data excluding $(\boldsymbol{x}_i, y_i)$
- This is known as **stacking**, which stands for "stacked generalization".