THE UNIVERSITY OF CHICAGO

STAT 37710 / CMSC 35400 / CAAM 37710 Machine Learning

Boosting

Cong Ma

AdaBoost for binary classification

We begin by describing the most popular boosting algorithm due to Freund and Schapire (1997) called "AdaBoost.M1." Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

and the expected error rate on future predictions is $\text{E}_{XY} I(Y \neq G(X))$.

- Purpose of Boosting: sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers

Weak learner to strong learner?

- 1988 Kearns and Valiant: "Can **weak learners** be combined to create a **strong learner?**"

Weak learner definition (informal):

An algorithm $\mathcal{A}$ is a *weak learner* for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $\mathcal{A}$ correctly classifies h with error at most $1/2 - \gamma$

- 1990 Robert Schapire: "Yup!"
- 1995 Schapire and Freund: "Practical for 0/1 loss" AdaBoost
- 2001 Friedman: "Practical for arbitrary losses"

Figure for AdaBoost

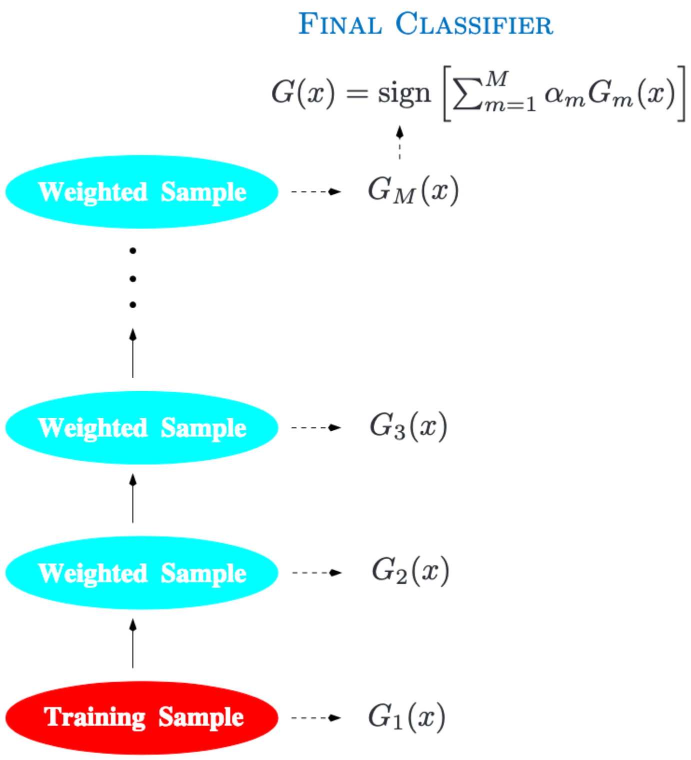

FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $\boxed{D_1(i) = 1/m \text{ for } i = 1, \ldots, m.}$ Initial Distribution of Data

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.

Train model

- Aim: select h_t with low weighted error:

$$\boxed{\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i].}$$

Error of model

- Choose $\boxed{\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)}$. Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$\boxed{D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}}$$

Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$\boxed{H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).}$$

Final average

Theorem: training error drops exponentially fast

http://rob.schapire.net/papers/explaining-adaboost.pdf

Boosting fits an additive model

The success of boosting is really not very mysterious. The key lies in expression (10.1). Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions. Here the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. More generally, basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{10.3}$$

where $\beta_m, m = 1, 2, \ldots, M$ are the expansion coefficients, and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x, characterized by a set of parameters γ. We discuss basis expansions in some detail in Chapter 5.

Typically these models are fit by minimizing a loss function averaged over the training data, such as the squared-error or a likelihood-based loss function,

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right). \tag{10.4}$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$.

Boosting for regression

$$L(y, f(x)) = (y - f(x))^2,$$

one has

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \end{aligned} \tag{10.7}$$

where $r_{im} = y_i - f_{m-1}(x_i)$ is simply the residual of the current model

AdaBoost with exponential loss

We now show that AdaBoost.M1 (Algorithm 10.1) is equivalent to forward stagewise additive modeling (Algorithm 10.2) using the loss function

$$L(y, f(x)) = \exp(-y\, f(x)). \tag{10.8}$$

For AdaBoost the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. Using the exponential loss function, one must solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x_i))]$$

for the classifier G_m and corresponding coefficient β_m to be added at each step. This can be expressed as

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i)) \tag{10.9}$$

with $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$. Since each $w_i^{(m)}$ depends neither on β

Why does boosting work?

- AdaBoost can be understood as a procedure for greedily minimizing the exponential loss over T rounds:

$$\ell(y_i, h(\mathbf{x}_i)) = \exp(-y_i h(\mathbf{x}_i)) \quad \text{where} \quad h(\mathbf{x}_i) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)$$

  - Why?

“Prediction games and arcing classifiers” [Leo Breiman, 1999]
https://www.stat.berkeley.edu/~breiman/games.pdf

Interpretation of Adaboost

- Choosing the first classifier

$$(\alpha_1, \hat{h}_1) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \ell(y_i, \alpha h(\mathbf{x}_i)) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \exp(-y_i \cdot \alpha h(\mathbf{x}_i))$$

- Update at round t

$$\tilde{h}_{t-1}(\mathbf{x}) = \sum_{\tau=1}^{t-1} \alpha_\tau \hat{h}_\tau(\mathbf{x})$$

$$\begin{aligned}(\alpha_t, \hat{h}_t) &= \arg\min_{\alpha,h} \sum_{i=1}^{m} \ell(y_i, \tilde{h}_{t-1}(\mathbf{x}) + \alpha h(\mathbf{x}_i)) \\ &= \arg\min_{\alpha,h} \sum_{i=1}^{m} \exp(-y_i \cdot (\tilde{h}_{t-1}(\mathbf{x}) + \alpha h(\mathbf{x}_i)))\end{aligned}$$

Interpretation of Adaboost

$$(\alpha_t, \hat{h}_t) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \exp(-y_i \cdot (\tilde{h}_{t-1}(\mathbf{x}_i) + \alpha h(\mathbf{x}_i)))$$

$$= \arg\min_{\alpha,h} \sum_{i=1}^{m} \underbrace{\exp(-y_i \tilde{h}_{t-1}(\mathbf{x}_i))}_{w_i^{(t)}} \exp(-y_i \cdot \alpha h(\mathbf{x}_i))$$

- Correcting the label for misclassified points
  - Giving those points higher weights when training classifier in future iterations

- We will solve h and α separately

Solving for h

- Fix α, $\hat{h}_t = \arg\min_h \sum_{i=1}^m w_i^{(t)} \exp(-y_i \cdot \alpha h(\mathbf{x}_i))$ $y_i \in \{-1, +1\}$.

$$= \arg\min_h \sum_{i: \mathbb{1}(h(x_i) = y_i)} w_i^{(t)} \exp(-\alpha) + \sum_{i: \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)} \exp(+\alpha)$$

$$= \arg\min_h \underline{e^{-\alpha} \cdot \sum_{i: \mathbb{1}(h(x_i) = y_i)} w_i^{(t)}} + e^{\alpha} \sum_{i: \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)} \underline{+ e^{-\alpha} \sum_{i: \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)}} - e^{-\alpha} \sum_{i: \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)}$$

$$= \arg\min_h e^{-\alpha} \underbrace{\sum_{i=1}^m w_i^{(t)}} + (e^{\alpha} - e^{-\alpha}) \cdot \sum_{i: \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)}$$

$$= \arg\min_h e^{-\alpha} + (e^{\alpha} - e^{-\alpha}) \cdot \frac{\sum_{i: \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)}}{\sum_{i=1}^m w_i^{(t)}}$$

16

[redacted]

$$\hat{h}_t = \underset{h}{\arg\min}\ e^{-\alpha} + \left(e^{\alpha} - e^{-\alpha}\right) \cdot \underbrace{\frac{\sum_{i:\, \mathbb{1}(h(x_i) \neq y_i)} w_i^{(t)}}{\sum_{i=1}^{m} w_i^{(t)}}}_{err_h^{(t)},\ \text{independent of } \alpha.}$$

Solving for α

- Now solve for α

$$\alpha_t = \operatorname*{argmin}_{\alpha} \underbrace{(e^{\alpha} - e^{-\alpha}) \cdot err_h^{(t)} + e^{-\alpha}}_{f_h(\alpha)}$$

$$\frac{\partial f_h(\alpha)}{\partial \alpha} = (e^{\alpha} + e^{-\alpha}) \cdot err_h^{(t)} - e^{-\alpha} = 0$$

$$\Rightarrow (e^{\alpha} + e^{-\alpha})\, err_h^{(t)} = e^{-\alpha}$$

$$\Rightarrow err_h^{(t)} = \frac{e^{-\alpha}}{e^{\alpha} + e^{-\alpha}}, \quad 1 - err_h^{(t)} = \frac{e^{\alpha}}{e^{\alpha} + e^{-\alpha}}$$

$$\Rightarrow e^{2\alpha} = \frac{1 - err_h^{(t)}}{err_h^{(t)}} \Rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - err_h^{(t)}}{err_h^{(t)}}$$

AdaBoost weight update

- Putting things together,

$$\hat{h}_t = \arg\min_h \underbrace{\frac{1}{\sum_{i=1}^m w_i^{(t)}} \sum_{i=1}^m w_i^{(t)} \mathbb{1}[h(\mathbf{x}_i) \neq y_i]}_{\mathrm{err}_{\hat{h}_t}} \qquad \alpha_t = \frac{1}{2} \ln \left(\frac{1 - \mathrm{err}_{\hat{h}_t}}{\mathrm{err}_{\hat{h}_t}} \right)$$

- Therefore, weights for next round are

$$\begin{aligned} w_i^{(t+1)} &= \exp(-y_i(\tilde{h}_{t-1}(\mathbf{x}) + \alpha_t \hat{h}_t(\mathbf{x})) \\ &= \underbrace{\exp(-y_i \tilde{h}_{t-1}(\mathbf{x}_i))}_{w_i^{(t)}} \cdot \exp(-\alpha_t y_i \hat{h}_t(\mathbf{x}_i)) \end{aligned}$$

Why do we care about exponential loss?

- Fisher consistent loss

It is easy to show (Friedman et al., 2000) that

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}, \qquad (10.16)$$

Gradient boosting

- Consider a generic loss function
  - E.g. squared loss, exponential loss

- Given current predictor $\tilde{h}_{t-1}(\mathbf{x})$, we aim to find new predictor h(x) so that the sum $\tilde{h}_{t-1}(\mathbf{x}) + h(\mathbf{x})$ pushes the loss towards its minimum as quickly as possible

- Gradient boosting: choose h in the direction of the negative gradient of the loss

Gradient boosting

- Fit a model to the negative gradients

- XGBoost is a python package for “extreme” gradient boosting
  - Folk wisdom: knowing logistic regression and XGBoost gets you 95% of the way to a winning Kaggle submission for most competitions
  - State-of-the-art prediction performance
    - Won Netflix Challenge
    - Won numerous KDD Cups
    - Industry standard

Gradient Boosting

start with an initial model, e.g. $\tilde{h}_1(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$

for $b = 1, 2, \dots$

calculate negative gradients

$$-g(x_i) = -\frac{\partial L(y_i, \tilde{h}_b(x_i))}{\partial h_b(x_i)}$$

fit a model h_b (e.g. tree) to negative gradients: $h_b = \underset{h}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(-g(x_i), h(x_i))$

$$\tilde{h}_{b+1}(x) = \tilde{h}_b(x) + \beta_b h_b(x)$$

where β_b is a step size parameter we find computationally to minimize the loss.

if $\tilde{h}_{b+1} \approx \tilde{h}_b$, STOP

Willett & Chen (2020). “CMSC 35400: Machine Learning”

References & acknowledgement

- Hastie et al. (2009). “The Elements of Statistical Learning”
  - Ch 10.1 , “Boosting Methods”
  - Ch 10.4, “Exponential Loss and AdaBoost”
- Willett & Chen (2020). “CMSC 35400: Machine Learning”
- Yue (2018). “Machine Learning & Data Mining”
  - Lecture 5, “Decision Trees, Bagging & Random Forests”
- Schapire (2013). “Explaining AdaBoost”
  - http://rob.schapire.net/papers/explaining-adaboost.pdf