Lecture 18: Model Stacking and Boosting

Statistical Learning and Data Mining

Xingye Qiao
Department of Mathematical Sciences
Binghamton University

E-mail: qiao@math.binghamton.edu

Read: ELSII Ch. 10, and ISLR 8.2

Outline

The next section would be . . . . . .

1. Model Stacking
2. Boosting

Motivation

- Random forest and bagging: combine the power of weak learners to obtain great predictive power. Here the learners are obtained by applying the same algorithm to different bootstrapped samples.
- Motivation of Model Stacking: Combine learners (regressors or classifiers) to improve the performance, where the learners may be of different types (e.g. can be kNN classifier, SVM and LDA)
- A weighted average of different types of base learners.

Example of stacking

- A total of M models: $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_M$.
- Proposed stacked model:

$$\sum_{m=1}^{M} w_m \hat{f}_m(x);$$

  How to find w_m's?
- $\hat{f}_m^{-i}(x)$ prediction at x, using model m, trained from the dataset with the observation i removed.
- The weights are found by

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \left[y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(x_i) \right]^2, \text{ with } w_m \geq 0 \text{ and } \sum_m w_m = 1$$

- Convex aggregation

- Find the best convex aggregation to improve the performance.
- Concern about overfitting? Built prevention mechanism into the process by the use of the leave-one-out sample.
- Close connection between stacking and model selection via leave-one-out cross-validation
- The weight can depend on x; hence a locally adaptive version of weighting.
- Leave-one-out is just one approach. Can use separate training set and stacking set (HOW?)

Another example of stacking

- Divide the data into training and stacking set. Use the training data to find the base learners. Use the stacking set to combine.
- The weights are found by

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{n_{st}} \left[y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{tr\cdot}(x_i) \right]^2 ,$$

$$\text{with } w_m \geq 0 \text{ and } \sum_m w_m = 1$$

- Question: what would happen if we use the same data for training and for stacking???

Stacking in practice

- Examples above are for regression problems. Stacking can also be applied to classification problems (the only difference is an appropriate loss function.)
- Simple way to stack models
  1. Train models using the training set
  2. Predict the stacking set using the M different models.
  3. Use the M predictions in the stacking set as inputs (M variables), train a final stacked model (using least square for regression; using logistic regression, SVM, random forest etc. for classification). But pay attention to the constraints for the weights.

Two Important Rules for Ensemble Learning

- Average. Averaging reduces variance.
- Randomize. Randomizing reduces correlation among ensemble members - closer to iid.

Recall random forest.

The next section would be . . . . . .

1. Model Stacking

2. Boosting
   - Boosting algorithm
   - Additive logistic model and exponential loss

The next section would be . . . . . .

Boosting

- To enhance accuracy of "weak" learner(s)
- Create a strong learner by substantially "boosting" the performance of each single weak learner.
- Focus on binary classification
- Here each weak learner must be slightly better than random guessing (error rate < 0.5).
- First really useful Boosting algorithm: Schapire (1990), Freund (1995), and **Schapire and Freund (1997)**

AdaBoost = Adaptive Boosting.

AdaBoost.M1

We introduce AdaBoost.M1 in the next slide. We follow the introduction in ESL (Chapter 10). The one in Izenman can be specialized to the case we will introduce here.

- Input: $\mathcal{D} := \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, n\}$ where $y_i \in \{+1, -1\}$

Core Ideas:

- A weaker learner is applied to an **adaptively weighted** training sample sequentially.
- Can show that this is equivalent to fitting to residuals.
- All resulting models are **weighted** in the end to form a final model.

Note that there are two types of weightings: one is for the observations; the other for the learners.

AdaBoost.M1

1. Initialize the weights $w_i = 1/n$ for $i = 1, \ldots, n$
2. For $t = 1, \ldots, T$
   a. Fit a classifier $\widehat{\phi}_t(\boldsymbol{x})$ to the data with weight vector $\{w_i\}$
   b. Compute the weighted training data prediction error:

$$E_t := \sum_{i=1}^{n} w_i \mathbb{1}_{\left\{y_i \neq \widehat{\phi}_t(\boldsymbol{x}_i)\right\}}$$

   c. Compute $\alpha_t := \log\left(\frac{1-E_t}{E_t}\right)$
   d. Update weights: $w_i \leftarrow \text{normalize}\{w_i \cdot e^{\alpha_t \mathbb{1}\left[y_i \neq \widehat{\phi}_t(\boldsymbol{x}_i)\right]}\}$
3. Final model:

$$\widehat{\phi}_{AdaBoost}(\boldsymbol{x}) := \text{sign}\left\{\sum_{t=1}^{T} \alpha_t \widehat{\phi}_t(\boldsymbol{x})\right\}$$

- The same learner is applied to an adaptively changing training data repeatedly.
  - Early version of AdaBoost allowed different algorithms
- $\alpha_t := \log\left(\frac{1-E_t}{E_t}\right)$ is a measure of fit for the model $\widehat{\phi}_t(\cdot)$ to the current weighted data. Perfect fit $=$'$\alpha_t \to \infty$'.
  - Hence, α_t becomes the weights for model $\widehat{\phi}_t(\cdot)$ in final model.
- Misclassified observations are weighted heavier [by a factor of $e^{\alpha_t} = \left(\frac{1-E_t}{E_t}\right)$], i.e., emphasizing more on them in next round.
  - If $E_t \approx 1/2$, then the magnitude of the change is milder.
- The normalize$\{c_i\}$ step makes the weights sum up to 1.
- Although weights for those correctly classified observations appears unchanged, they turn smaller due to normalization.

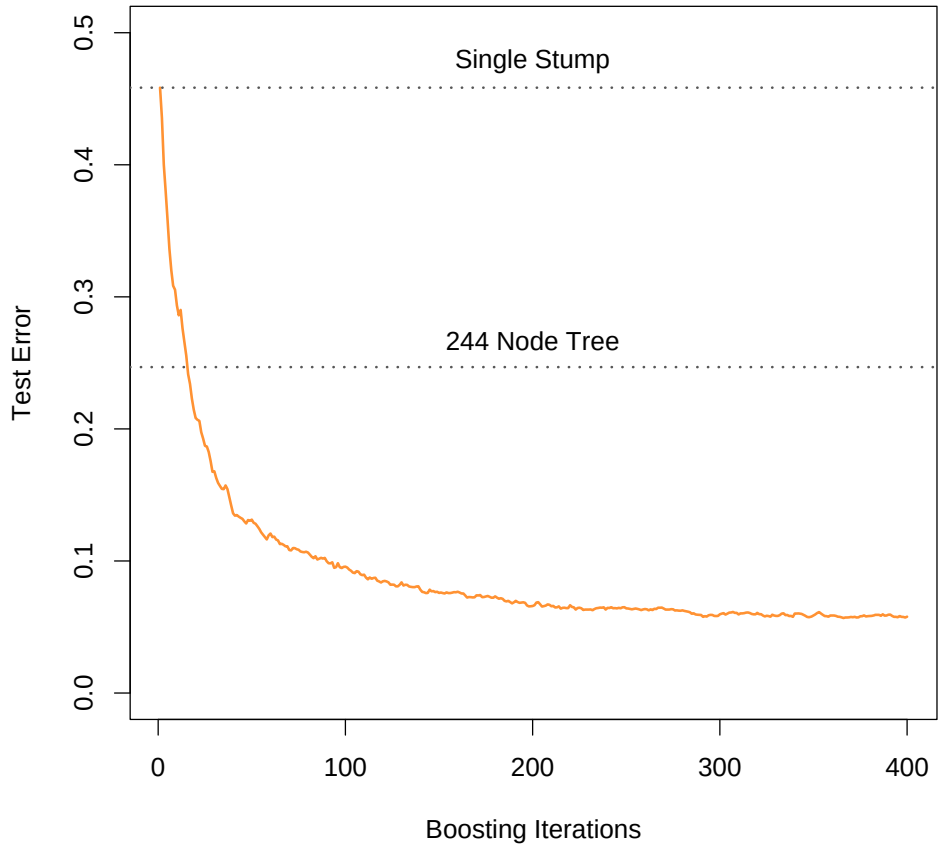

FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

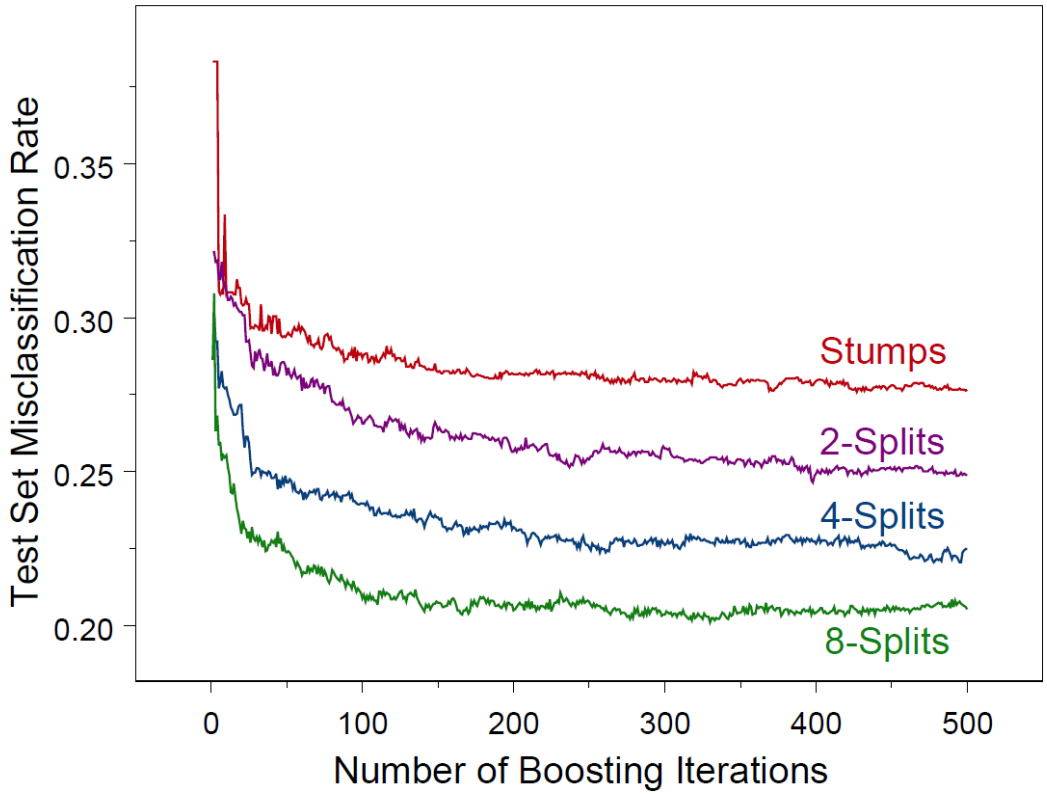

Figure: After 100 iterations, the test set misclassification rates are stablized.

- Empirical evidence has shown that Boosting seems to be resistant to overfit: the testing set misclassification rate will be stable and not go up as iteration grows.
- Some other empirical experiments have shown that the overfitting may appear, if iterations go for a very long time.
- Leo Breiman once speculated that the generalization error of boosting may eventually go to the Bayes error (the theoretically best one can do.)
- Many attempts have been made to try to explain the mysteriously good performance of Boosting.

In the next subsection, we provide a statistical interpretation of boosting. See ESL 10.2 – 10.5 and Izenman 14.3.6 for more details. This view was initially given by Friedman, Hastie and Tibshirani (2000), "Additive logistic regression: a statistical view of boosting," *the Annals of Statistics*

The next section would be . . . . . .

We first introduce additive model and exponential loss. We then show that AdaBoost is equivalent to fitting an additive model with exponential loss.

- An additive model takes the form

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m g(\boldsymbol{x}; \gamma_m)$$

  where β_m is the weight for the mth model and γ_m is the estimated parameters for the mth model.
- One needs to fit β_m and γ_m simultaneously.
- Goal is to minimize $\sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M} \beta_m g(\boldsymbol{x}_i; \gamma_m))$ for an appropriate loss function.
- Such optimization can be quite complicated, even if the loss function has a simple form. One possible way to solve this is to sequentially estimate (β_m, γ_m) for $m = 1, \ldots, M$

Forward stagewise additive model

1. Initialize $f_0(\boldsymbol{x}) = 0$ (no model)
2. For $m = 1$ to M
   a. Estimate

$$(\beta_m, \gamma_m) := \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L[y_i, f_{m-1}(\boldsymbol{x}_i) + \beta g(\boldsymbol{x}_i; \gamma)]$$

   b. Set $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m g(\boldsymbol{x}; \gamma_m)$

- After each iteration the model is enhanced.
- At each iteration, the optimization involves only one pair of (β_m, γ_m)

Exponential loss

- For binary classification problems, we call $u := yf(\boldsymbol{x})$ a functional margin. The larger the margin is, the more confidence we have that the data point $(\boldsymbol{x}, y)$ is correctly classified.
- We consider the exponential loss

$$L(y, f(\boldsymbol{x})) := e^{-yf(\boldsymbol{x})}$$

- The risk function is $\mathrm{E}(e^{-Yf(\boldsymbol{X})})$. In order to find a f that minimizes $\mathrm{E}(e^{-Yf(\boldsymbol{X})})$, we only need to minimize $\mathrm{E}(e^{-Yf(\boldsymbol{X})}|\boldsymbol{X} = \boldsymbol{x})$ for any $\boldsymbol{x}$.
- Write

$$\begin{aligned}\ell(f(\boldsymbol{x})) := \mathrm{E}(e^{-Yf(\boldsymbol{X})}|\boldsymbol{X} = \boldsymbol{x}) \\ =& e^{-f(\boldsymbol{x})}P(Y = 1|\boldsymbol{X} = \boldsymbol{x}) + e^{f(\boldsymbol{x})}P(Y = -1|\boldsymbol{X} = \boldsymbol{x}) \\ =& e^{-f(\boldsymbol{x})}\eta(\boldsymbol{x}) + e^{f(\boldsymbol{x})}[1 - \eta(\boldsymbol{x})]\end{aligned}$$

Take the derivative of $\ell(f(\boldsymbol{x}))$ with respect to $f(\boldsymbol{x})$, and set it to 0, we have

$$\begin{aligned} 0 = \ell'(f(\boldsymbol{x})) &= -e^{-f(\boldsymbol{x})}\eta(\boldsymbol{x}) + e^{f(\boldsymbol{x})}[1-\eta(\boldsymbol{x})] \\ \Rightarrow e^{-f(\boldsymbol{x})}\eta(\boldsymbol{x}) &= e^{f(\boldsymbol{x})}[1-\eta(\boldsymbol{x})] \\ \Rightarrow e^{2f(\boldsymbol{x})} &= \frac{\eta(\boldsymbol{x})}{1-\eta(\boldsymbol{x})} \\ \Rightarrow 2f(\boldsymbol{x}) &= \log\left(\frac{\eta(\boldsymbol{x})}{1-\eta(\boldsymbol{x})}\right) \\ \Rightarrow f(\boldsymbol{x}) &= \frac{1}{2}\log\left(\frac{\eta(\boldsymbol{x})}{1-\eta(\boldsymbol{x})}\right) \end{aligned}$$

Recall that this gives the model for logistic regression (when f is a linear function of $\boldsymbol{x}$).

Incidentally, this minimizer of the population risk function under exponential loss coincides with that under the logistic loss function

$$L(y, f(\boldsymbol{x})) := \log(1 + e^{-yf(\boldsymbol{x})})$$

AdaBoost = A.M. + Exponential Loss

- We combine the forward stagewise additive modelling with the exponential loss.
- Consider that the basis $g(\cdot)$ in the additive model is the weak learner $\widehat{\phi}(\cdot)$ in AdaBoost. What this weak learner really is does not matter.
- In this setting, at each iteration, we seek to find

$$\begin{aligned}
&\underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} e^{-y_i \cdot [f_{m-1}(\boldsymbol{x}_i) + \beta \widehat{\phi}(\boldsymbol{x}_i;\gamma)]} \\
&= \sum_{i=1}^{n} e^{-[y_i \cdot f_{m-1}(\boldsymbol{x}_i) + y_i \beta \widehat{\phi}(\boldsymbol{x}_i;\gamma)]} \\
&= \sum_{i=1}^{n} w_i^{(m)} e^{-y_i \beta \widehat{\phi}(\boldsymbol{x}_i;\gamma)}
\end{aligned}$$

where $w_i^{(m)} := e^{-y_i \cdot f_{m-1}(\boldsymbol{x}_i)}$

When minimizing $\sum_{i=1}^{n} w_i^{(m)} e^{-y_i \beta \widehat{\phi}(\boldsymbol{x}_i;\gamma)}$, we first fix β:

$$
\begin{aligned}
&\sum_{i=1}^{n} w_i^{(m)} e^{-\beta y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)} \\
=&\sum_{y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=1} w_i^{(m)} e^{-\beta} + \sum_{y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=-1} w_i^{(m)} e^{\beta} \\
=&e^{-\beta} \sum_{y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=1} w_i^{(m)} + e^{\beta} \sum_{y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=-1} w_i^{(m)} \\
=&e^{-\beta} \sum_i w_i^{(m)} \mathbb{1}_{\left[y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=1\right]} + e^{\beta} \sum_i w_i^{(m)} \mathbb{1}_{\left[y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=-1\right]} \\
=&\left(e^{\beta} - e^{-\beta}\right) \boxed{\textstyle\sum_i w_i^{(m)} \mathbb{1}_{\left[y_i \widehat{\phi}(\boldsymbol{x}_i;\gamma)=-1\right]}} + e^{-\beta} \sum_i w_i^{(m)}
\end{aligned}
$$

Hence $\widehat{\phi}_m(\boldsymbol{x})$ minimizes $\dfrac{\sum_i w_i^{(m)} \mathbb{1}_{\left[y_i \neq \widehat{\phi}_m(\boldsymbol{x}_i)\right]}}{\sum_i w_i^{(m)}}$, the weighted 0-1 loss, i.e. E_m

Once we have $\widehat{\phi}_t(\cdot)$, we find the minimizing β:

$$\ell(\beta) := \left(e^{\beta} - e^{-\beta}\right) \boxed{E_m} + e^{-\beta}$$

$$\ell'(\beta) := \left(e^{\beta} + e^{-\beta}\right) \boxed{E_m} - e^{-\beta} = 0$$

which is minimized at

$$\beta_m = \frac{1}{2} \log\left(\frac{1 - E_m}{E_m}\right)$$

The additive model continues with $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m \widehat{\phi}_m(\boldsymbol{x})$.
This means that the weight for the next iteration is
$w_i^{(m+1)} = e^{-y_i \cdot f_m(\boldsymbol{x}_i)} = e^{-y_i \cdot [f_{m-1}(\boldsymbol{x}_i) + \beta_m \widehat{\phi}_m(\boldsymbol{x}_i)]} = w_i^{(m)} e^{-y_i \beta_m \widehat{\phi}_m(\boldsymbol{x}_i)}$

Multiplicative factor on the weight updating is

$$e^{-\beta_m y_i \widehat{\phi}_m(\boldsymbol{x}_i)}$$

Recall that $y_i, \widehat{\phi}_m(\boldsymbol{x}_i) \in \{+1, -1\}$. So

$$-y_i \widehat{\phi}_m(\boldsymbol{x}_i) = 2\mathbb{1}_{\left\{y_i \neq \widehat{\phi}_m(\boldsymbol{x}_i)\right\}} - 1$$

Hence

$$\begin{aligned} e^{-\beta_m y_i \widehat{\phi}_m(\boldsymbol{x}_i)} &= e^{\beta_m (2\mathbb{1}_{\left\{y_i \neq \widehat{\phi}_m(\boldsymbol{x}_i)\right\}} - 1)} \\ &= e^{2\beta_m \mathbb{1}_{\left\{y_i \neq \widehat{\phi}_m(\boldsymbol{x}_i)\right\}}} e^{-\beta_m} \\ &= e^{\alpha_m \mathbb{1}_{\left\{y_i \neq \widehat{\phi}_m(\boldsymbol{x}_i)\right\}}} e^{-\beta_m} \end{aligned}$$

where $\alpha_m = \log\left(\frac{1-E_m}{E_m}\right)$ was defined in the AdaBoost.M1 algorithm. Note that $e^{-\beta_m}$ will vanish due to normalization.

Short summary,

- The final model is the weighted sum of $\widehat{\phi}_m(\boldsymbol{x})$ with $\beta_m = \frac{1}{2}\alpha_m$ as weights (half of the weights in AdaBoost.M1)
- The observation weight is updated by $w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m \mathbb{1}\left\{y_i \neq \widehat{\phi}_m(\boldsymbol{x}_i)\right\}}$, the same as in AdaBoost.M1.
- In AdaBoost.M1, the weak learner might have its own loss function. But in principal, it should try to minimize the weighted 0-1 loss.

These suggest that AdaBoost.M1 is **approximately** solving an additive model with some weak learners as basis, with the forward stagewise algorithm as the optimization approach, and with the exponential loss as the loss function.

The reason that it is "approximately" solving, not "exactly" solving, is that the weak learner may not directly minimizes the weighted 0-1 loss.

LogitBoost

- Have shown that AdaBoost = Exponential loss (defined as $\exp(-yf(\boldsymbol{x}))$) plus Additive Modelling
- What if we use a different loss function from the exponential loss?
- LogitBoost minimizes the logistic loss (binomial deviance loss)

$$\log(1 + \exp(-yf(\boldsymbol{x})))$$

- How about the square error loss? - equivalent to regressing on the residuals.