Boosting

Ryan Miller

Grinnell College
Statistics

Introduction

- Random Forests use *bagging* to build an ensemble of decision tree models
  - Many trees trained on slightly different data contribute to the model's predictions
  - This is an *aggregation* approach, as each base model is *trained separately* and the predictions are aggregated

Introduction

- Random Forests use *bagging* to build an ensemble of decision tree models
  - Many trees trained on slightly different data contribute to the model's predictions
  - This is an *aggregation* approach, as each base model is *trained separately* and the predictions are aggregated
- Today we will discuss *boosting* approaches, where the base models in the ensemble are *trained sequentially*
  - Boosting was originally developed as a classifier aimed at combining many "weak" classifiers into a more powerful "committee" by Freund and Schapire (1997) as "Adaptive Boosting" or "AdaBoost.M1"

Bagging vs. Boosting

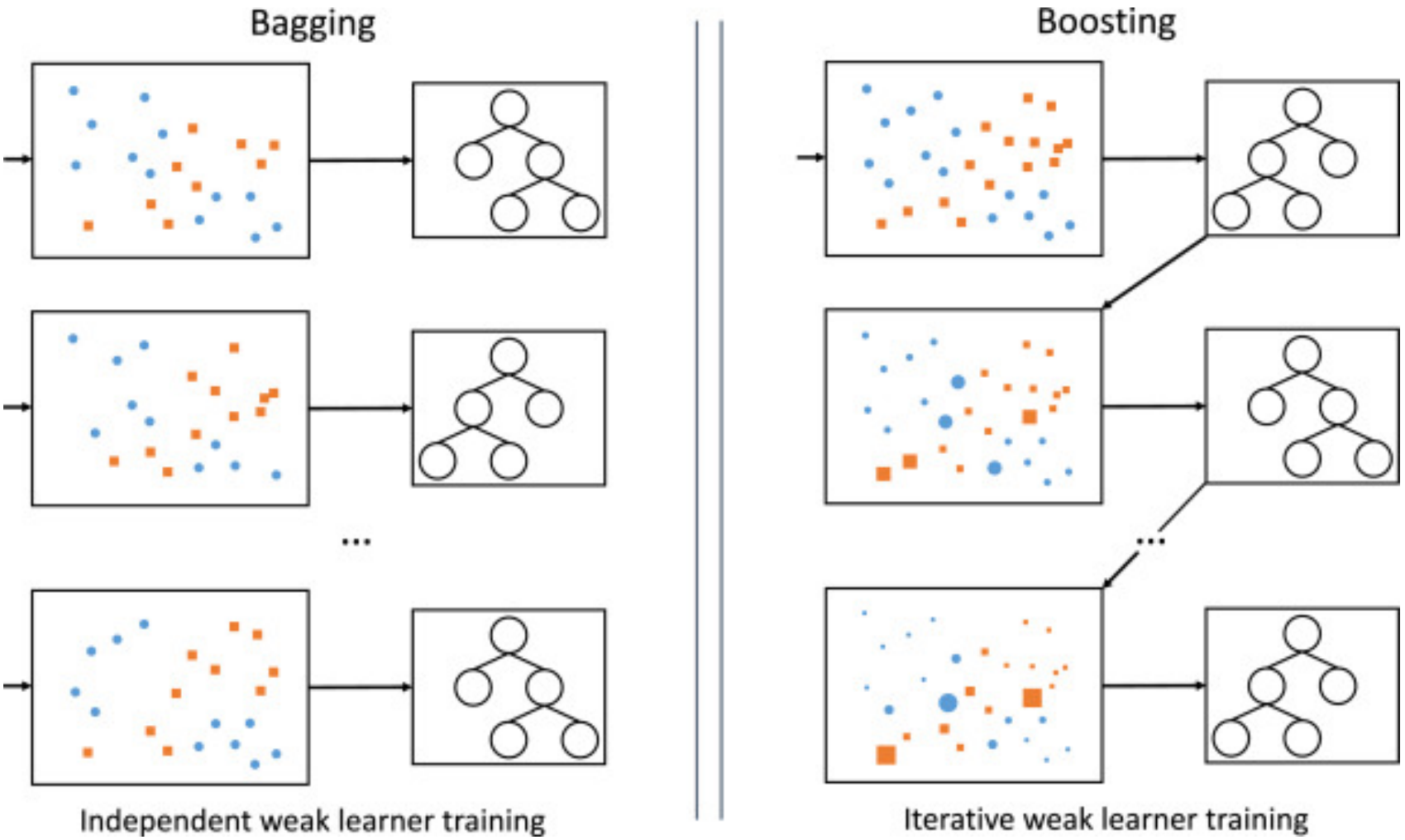

Image Credit: https://www.sciencedirect.com/science/article/pii/S1566253520303195

AdaBoost

To begin, suppose Y is a binary variable encoded by $\{-1, 1\}$, and define the error rate as:

$$\text{error} = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq G(\mathbf{x}_i))$$

- Here, $G(\mathbf{x}_i)$ represents the predicted class for an observation with predictors $\mathbf{x}_i$
  - Notice that if y_i does not match the predicted class the summation increases by 1
  - Thus, we see that error $= 1 -$ classification accuracy

AdaBoost

In AdaBoost, $G()$ is a sum of M sequentially built classifiers trained on differently weighted versions of the data:

$$G(x) = \text{Sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x) \right)$$

- $\alpha_1, \ldots, \alpha_M$ allow each classifier to contribute differently to the final prediction (thereby allowing some models to contribute more)
- Each training data-point, $(\mathbf{x}_i, y_i)$ is also given a different weight, w_{im}, at each iteration
  - At the first step of the algorithm, these weights are set to $\frac{1}{n}$, so that all observations contribute equally
  - At step m, the weights of observations misclassified by $G_{m-1}(x)$ are increased by a factor of $exp(\alpha_m)$

AdaBoost (algorithm)

Pseudocode for the original AdaBoost algorithm:

```
w = 1/n  # initialize weights
for i in 1:M:
  G_m = model.fit(X, w, y) # fit using weighted data
  err_m = error(G_m)       # calculate error
  alpha_m = log((1-err_m)/err_m)
  w_i = w_i*exp(a_m*(y_i != G_m(x_i)))  # reweight
```

- Each model's contribution in the ensemble is based upon it's accuracy (notice $log((1-0.5)/0.5) = 0$)
- If an observation was misclassified, its weight in the next model is increased by a factor of $exp(\alpha_m)$

AdaBoost (diagram)

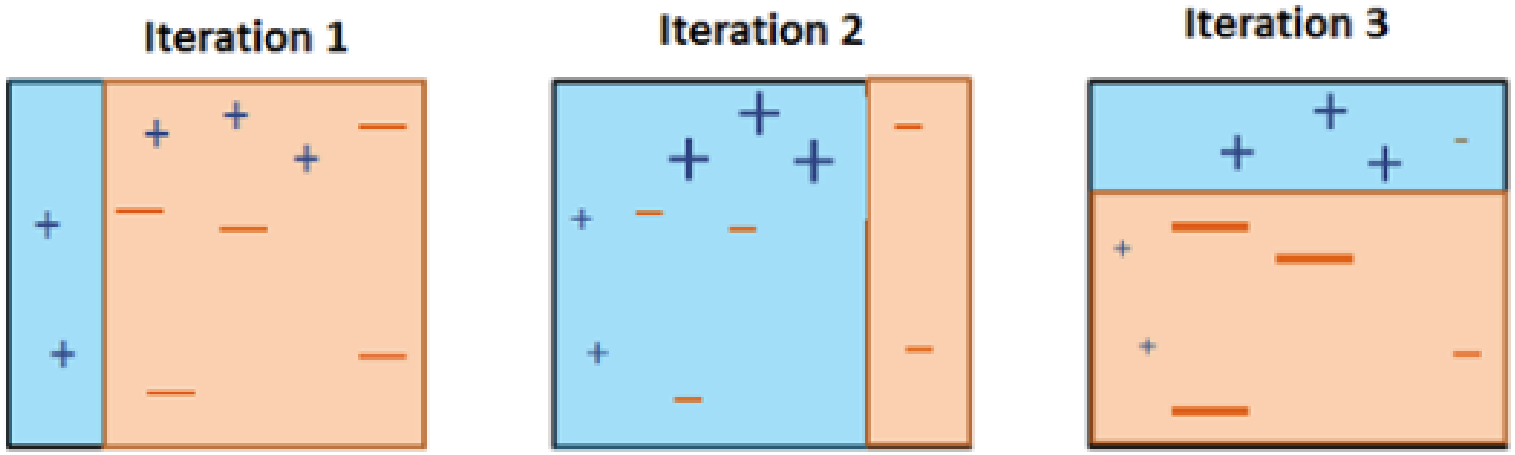

Image Credit: Packt Big data and Business Intelligence

Gradient Boosting

- AdaBoost re-weights observations that are misclassified before training the next model
  - Gradient boosting takes a more general approach - train subsequent models to the *residuals* (or loss contributions for cost functions other than squared error)
- At each iteration, t, of gradient boosting, the algorithm finds a base model (estimates $\hat{f}_t$):

$$\hat{f}_t(\mathbf{X}) = \arg\min_{f_t}\left(L(y, \hat{y}_{t-1} + \alpha f_t(\mathbf{X}))\right)$$

- For the squared error cost function, this is amounts to fitting the base model to the residuals

Gradient Boosting

Consider squared error as the objective function:

$$\frac{1}{n}\left(\mathbf{y} - (\hat{\mathbf{y}}_{t-1} + \alpha f_t(\mathbf{X}))\right)^T \left(\mathbf{y} - (\hat{\mathbf{y}}_{t-1} + \alpha f_t(\mathbf{X}))\right)$$

If we disregard terms without f_t, we have:

$$\frac{1}{n}(2\alpha \mathbf{y}^T f_t(X) - 2\alpha \hat{\mathbf{y}}_{t-1} f_t(X) + \alpha^2 f_t(X)^T f_t(X))$$

After differentiating WRT $f_t(X)$:

$$\frac{2\alpha}{n}(\mathbf{y} - \hat{\mathbf{y}}_{t-1} + \alpha f_t(X))$$

If $\alpha = 1$ and we substitute $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}_{t-1}$, we have $\frac{1}{n}(\mathbf{r} - f_t(\mathbf{X}))$, which is minimized when f_t is fit to the residuals.

Gradient Boosting (pseudocode)

```
r = y ## Initialize residuals
f = 0 ## Initialize model to zero

for i in 1:M:
    f_m = model.fit(X, r) ## Fit model to residuals
    f = f + alpha*f_m  ## Add new model to ensemble
    r = r - alpha*f_m  ## Update residuals
```

- The output is f, the ensemble model consisting of M different base models.
- The learning rate, α, is a small positive number that controls how quickly boosting learns (by limiting how much a model can contribute to the ensemble)

Gradient Boosting vs AdaBoost (diagram)

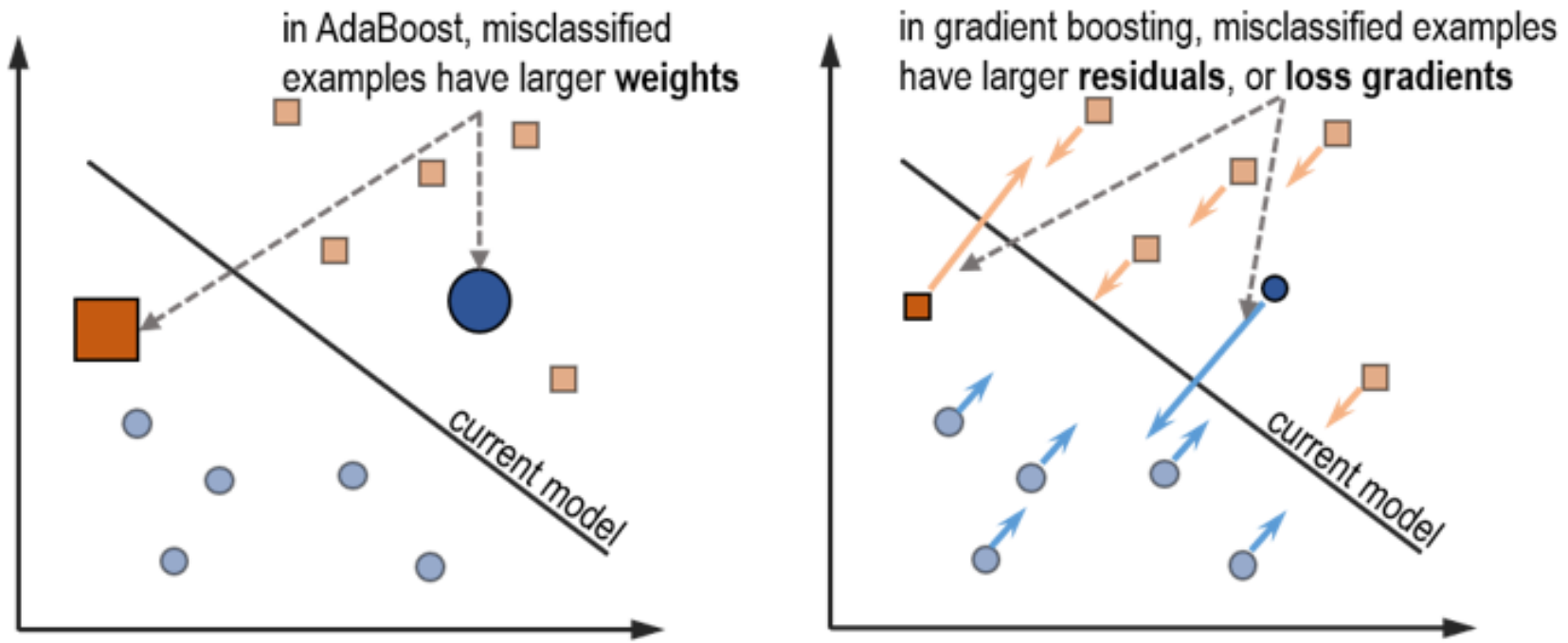

Image credit: Ensemble Methods for Machine Learning (textbook)

Gradient Boosting (comments)

- Unlike bagging (random forests), boosting can overfit if too many base models are used, so the number of boosting iterations should be careful chosen.
  - The learning rates (value of α) and the number of boosting iterations work jointly to dictate whether a model ends up being overfit or underfit
- Like random forests, boosting tends to work best using shallow simple decision trees (max depths of either 1 or 2)