# Contents

1 Boosting .................................................. 2

2 AdaBoost .................................................. 2
   2.1 Adaboost Algorithm ..................................... 4
   2.2 R package ada ............................................. 7

3 Gradient Boosting ........................................... 8
   3.1 $L_2$ Boosting .............................................. 8
   3.2 GBM (Gradient Boosting Machine) ......................... 10
   3.3 xgboost (Extreme Gradient Boosting) ....................... 11
   3.4 CatBoost .................................................. 13
   3.5 LightGBM ................................................. 13
1 Boosting

Boosting is a *sequential* ensemble method.

**Boosting Sketch**

- There are two main versions of boosting:
  - *Gradient Boosting*: fits the next model in the sequence $\hat{g}_k(x)$ to the (pseudo) residuals calculated from the predictions on the previous models $\sum_{l=0}^{k-1} \hat{w}_l \hat{g}_l(x)$.
  - *AdaBoost*: fits the next model to sequentially *weighted* observations. The weights are proportional to the how poorly the current models predict the observation.

- Boosting is primarily a *bias reducer*
  - The base models are often simple/weak (low variance, but high bias) models (like shallow trees)

2 AdaBoost

AdaBoost was motivated by the idea that many *weak* learners can be combined to produce a *strong* aggregate model.

- AdaBoost is for binary classification problems
- Trees are a popular base learner
  - *Weak* learners are usually used. For trees, this means shallow depth.
- At each iteration, the current model is evaluated.
  - The *ensemble weight* of model $m$ is based on its performance (on all the training data)
  - The *observation weight* of observation $i$ is increased if it is mis-classified and decreased if it is correctly classified.
– Thus, at each iteration, those observations that are mis-classified are weighted higher and get extra attention in the next iteration.

• Because Adaboost uses hard-classifiers, it is sensitive to unbalanced data and unequal misclassification costs.
  – Because the thresholds are set to $p > .50$
  – There are, of course, ways to account for unbalance and unequal costs in the algorithm
  – An improvement to AdaBoost, LogitBoost explicitly attempts to estimate the class probability during each iteration which will allow easier post-fitting adjustments for unequal costs
2.1 Adaboost Algorithm

Algorithm: AdaBoost

Inputs:
- \( D = \{(x_i, y_i)\}_{i=1}^n \), where \( y_i \in \{-1, 1\} \)
- Tuning parameters for base model \( \hat{g} \)

Algorithm:
1. Initialize observation weights \( w_i = 1/n \) for all \( i \)
2. For \( k = 1 \) to \( M \):
   a. Fit a classifier \( \hat{g}_k(x) \) that maps \((x_i, w_i)\) to \( \{-1, 1\} \). In other words, the classifier must make a hard classification using weighted observations.
   b. Compute the weighted mis-classification rate
      \[
      e_k = \frac{\sum_{i=1}^n w_i \mathbb{1}(y_i \neq \hat{g}_k(x_i))}{\sum_{i=1}^n w_i}
      \]
   c. Calculate the coefficient for model \( k \) (ensemble weight)
      \[
      a_k = \log \left( \frac{1 - e_k}{e_k} \right)
      \]
   d. Update the observations weights. Increase weights for observations that are mis-classified by model \( \hat{g}_k \) and decrease weights for the correctly classified observations.
      \[
      \tilde{w}_i = w_i \cdot \exp \left( a_k \cdot \mathbb{1}(y_i \neq \hat{g}_k(x_i)) \right)
      \]
      \[
      w_i = \frac{\tilde{w}_i}{\sum_{j=1}^n \tilde{w}_j} \quad \text{(re-normalize weights)}
      \]
3. Output final ensemble \( \hat{f}_M(x) \in [-1, 1] \)
   \[
   \hat{f}_M(x) = \sum_{k=1}^M a_k \hat{g}_k(x)
   \]
   - Or remap to a probability \( \hat{p}(x) = \frac{e^{\hat{f}_M(x)}}{1 + e^{\hat{f}_M(x)}} \)
2.1.1 Illustration with Stumps (depth = 1, n.nodes=2)
2.1.2 Illustration with depth = 2, n.nodes=4

![Diagram of decision trees with depth 2 and n.nodes=4](image-url)
2.2 R package ada

The R package ada provides an implementation of AdaBoost (and related methods).


  - \{Discrete, Real, Gentle\} AdaBoost
  - Logitboost
3 Gradient Boosting

In gradient boosting, instead of re-weighting the observations, each new model is fit to the functional gradients (i.e., a type of residuals)

- Gradient Boosting can fit to a variety of loss functions by simply changing how the residuals are calculated.
- Again, trees are often used as the base learner
  - For gradient boosting, regression trees are used

3.1 $L_2$ Boosting

$L_2$ boosting is based on the the squared error loss function

$$L(y_i, \hat{f}(x_i)) = \frac{1}{2}(y_i - \hat{f}(x_i))^2$$

- The residuals are

$$r_i = \left[ \frac{\partial L(y_i, f_i)}{\partial f_i} \right]_{f_i=\hat{f}(x_i)}$$

$$= y_i - \hat{f}(x_i)$$

- This is basically just re-fitting to the residuals.

**Algorithm: $L_2$ Boosting**

1. Initialize $\hat{f}_0(x) = \bar{y}$

2. For $k = 1$ to $M$:
   a. Calculate residuals $r_i = y_i - \hat{f}_{m-1}(x_i)$ for all $i$
   b. Fit a base learner (e.g., regression trees) to the residuals $\{(x_i, r_i)\}_{i=1}^n$ to get the model $\hat{g}_m(x)$
   c. Update the overall model $\hat{f}_m(x) = f_{m-1}(x) + \nu \hat{g}_m(x)$
      - $0 \leq \nu \leq 1$ is the step-size (shrinkage)

3. Final model is $\hat{f}_M(x) = \bar{y} + \sum_{k=1}^M \nu \hat{g}_k(x)$

- Like AdaBoost, emphasis is given to observations that are predicted poorly (large residuals)
3.1.1 Illustration using stumps (depth=1, n.nodes=2)
3.2 GBM (Gradient Boosting Machine)

- R package `gbm`
- GBM Documentation

3.2.1 Model/Tree Tuning Parameters

- Tree depth (`interaction.depth`)
  - Grows trees to a depth specified by `interaction.depth` (unless there are not enough observations in the terminal nodes)

- Minimum number of observations allowed in the terminal nodes (`n.minobsinnode`)

- Sub-sampling (`bag.fraction`)
  - *Stochastic Gradient Boosting*
  - Sample (without replacement) at each iteration

- Loss Function (`distribution`)
  - The loss function is determined by the `distribution` argument
  - Use `distribution="gaussian"` for squared error
  - Other options are: `bernoulli` (for logistic regression), `poisson` (for Poisson regression), `pairwise` (for ranking/LambdaMart), `adaboost` (for the adaboost exponential loss), etc.

3.2.2 Boosting Tuning Parameters

- Number of iterations/trees (`n.trees`)
  - Use cross-validation (or out-of-bag) to find optimal value
  - Can use the helper function `gbm.perf()` to get the optimal value
• Shrinkage parameter (shrinkage)
  – Set small, but the smaller the shrinkage, the more iterations/trees need to be used
  – “Ranges from 0.001 to 0.100 usually work”
• Cross-validation (cv.folds)
  – gbm has a built in cross-validation
  – no way to manually set the folds

3.2.3 Computational Settings

• Number of Cores (n.cores)
  – Only used when cross-validation is implemented

3.3 xgboost (Extreme Gradient Boosting)

• R package xgboost
• xgboost Documentation
• xgboost Paper

3.3.1 Model/Tree Tuning Parameters

• Different base leaners (booster)
  – gbtree is a tree
  – gblinear creates a (generalized) linear model (forward stagewise linear model)
• Tree building (tree_method)
  – To speed up the fitting, only consider making splits at certain quantiles of the input vector (rather than considering every unique value)
• Sub-sampling (subsample)
  – Stochastic Gradient Boosting
  – Sample (without replacement) at each iteration
• Feature sampling (colsample_bytree, subsample_baselevel, colsample_bynode)
  – Like used in Random Forest, the features/columns are subsampled
  – Can use a subsample of features for each tree, level, or node

Model Complexity Parameters

• Tree depth (max_depth)
  – Grows trees to a depth specified by max_depth (unless there are not enough observations in the terminal nodes)
  – Trees may not reach max_depth if the gamma or min_child_weight arguments are set.
• Minimum number of observations (or sum of weights) allowed in the terminal nodes (min_child_weight)
• Pruning (gamma or min_split_loss)
  – Minimum loss reduction required to make a further partition on a leaf node of the tree
The larger gamma is, the more conservative the algorithm will be

- ElasticNet type penalty (\(\lambda\) and \(\alpha\))
  - \(\lambda\) is an \(L_2\) penalty
  - \(\alpha\) is an \(L_1\) penalty

- Recall that trees model the response as a constant in each region
  \[
  \hat{f}_T(x) = \sum_{m=1}^{M} \hat{c}_m \mathbb{1}(x \in \hat{R}_m)
  \]

- Cost-complexity pruning found the optimal tree as the one that minimized the penalized loss objective function:
  \[
  C_\gamma(T) = \sum_{m=1}^{\lfloor T \rfloor} \text{Loss}(T) + \gamma |T|
  \]

- XGBoost selects a tree at each iteration using the following penalized loss:
  \[
  C_{\gamma,\lambda,\alpha}(T) = \sum_{m=1}^{\lfloor T \rfloor} \text{Loss}(T) + \gamma |T| + \frac{\lambda}{2} \sum_{m=1}^{\lfloor T \rfloor} \hat{c}_m^2 + \alpha \sum_{m=1}^{\lfloor T \rfloor} |\hat{c}_m|
  \]

- Loss Function (objective)
  - The loss function is determined by the \texttt{objective} argument
  - Use \texttt{reg:squarederror} for squared error
  - Other options are: \texttt{reg:logistic} or \texttt{binary:logistic} (for logistic regression), \texttt{count:poisson} (for Poisson regression), \texttt{rank:pairwise} (for ranking/LambdaMart), etc.

### 3.3.2 Boosting Tuning Parameters

- Shrinkage parameter (\texttt{eta} or \texttt{learning_rate})
  - Set small, but the smaller the \texttt{eta}, the more iterations/trees need to be used

- Number of iterations/trees (\texttt{num_rounds})
  - Use cross-validation (or out-of-bag) to find optimal value

- Cross-validation (\texttt{xgb.cv})
  - \texttt{xgboost} has a built in cross-validation
  - It is possible to manually set the folds

### 3.3.3 Computational Settings

- Number of Threads (\texttt{nthread})

  - Used for finding tree split points and evaluating/calculating the loss function
3.4 CatBoost

- CatBoost Documentation
- Model/Tree Tuning Parameters:

3.5 LightGBM

- R Package: [https://github.com/microsoft/LightGBM/tree/master/R-package](https://github.com/microsoft/LightGBM/tree/master/R-package)
- LightGBM Documentation
- Model/Tree Tuning Parameters:
• Boosting Tuning Parameters: