Ensemble Learning part I: Bagging

COMP 551 Applied Machine Learning

Isabeau Prémont-Schwarz
School of Computer Science
McGill University

Fall 2024

Please complete the online evaluation for the course on Mercury.

https://go.blueja.io/qLhY92bmaEW4w8hLQ53-LA

Your feedback is very important to us.

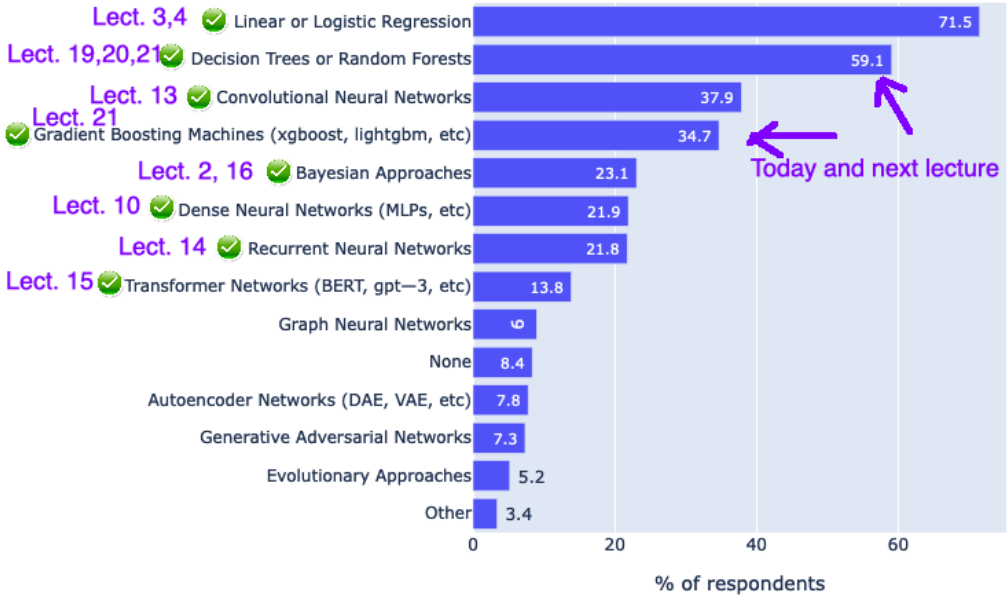
Thank you!

Outline

Acknowledgement

- The slides are adapted from Prof. Yue Li's slides which are based on Chapter 18.2 from the Murphy 2022 complemented by Bishop06 Chapter 14.3 and Prof. Reihaneh Rabbany's slides.

Most commonly used ML algorithms in Kaggle 2022 survey



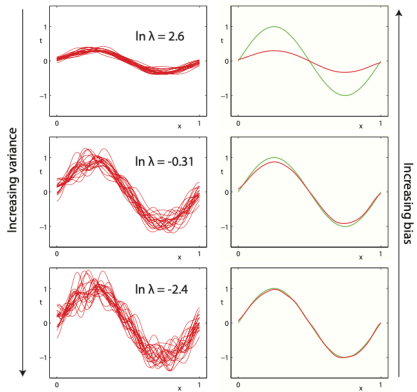
Learning objectives

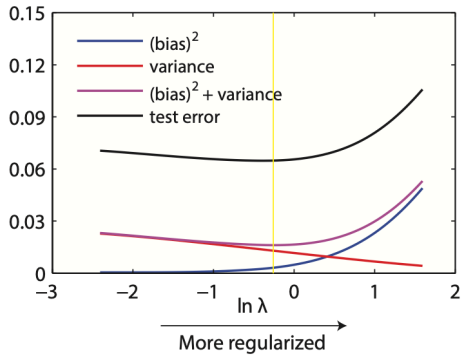
- Advantages of ensemble learning
- Bootstrap aggregation (Bagging)
- Random Forest
- Boosting
  - Least square additive boosting
  - GradientBoosting

Recall Generalization Lectures and A2 : bias-variance trade-off

Bias-variance trade-off

$$\mathbb{E}_{\mathcal{D}}\left[(\hat{f}(\mathbf{x};\mathcal{D})-y)^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(\mathbf{x})-\mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right)^2\right]}_{\textit{Variance}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(\mathbf{x})]-f(\mathbf{x})\right)^2\right]}_{\textit{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\epsilon^2\right]}_{\textit{noise}}$$

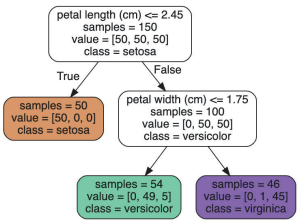




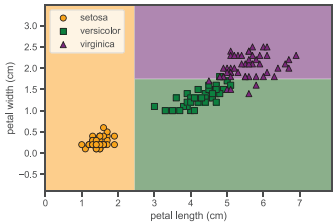
Outline

Decision trees are sensitive to the change of training data

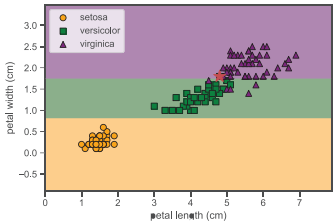
DT (tree depth = 2) for Iris flower



Decision surface of the DT

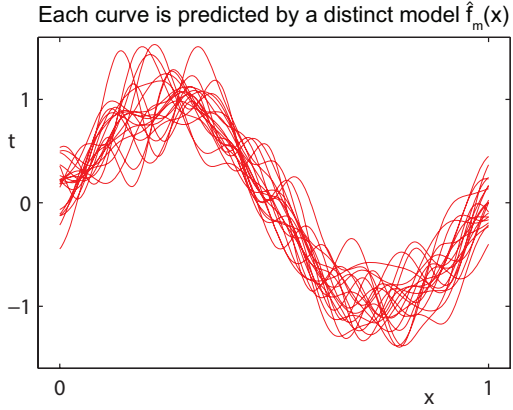


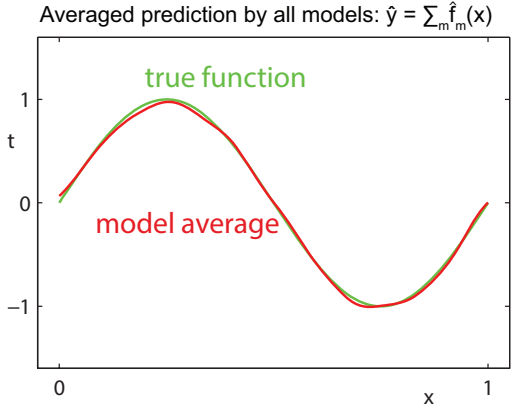
Fit to the same data but omit a single data point (shown by star)



Decision trees are unstable: small changes to the input data can have large effects on the structure of the tree, due to the greedy and hierarchical nature of the tree-growing process.

Recall Generalization Lectures & A2: model averaging reduces variance





$$\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}(\mathbf{x};\mathcal{D}) - y\right)^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right)^2\right]}_{\textit{Variance}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(\mathbf{x})] - f(\mathbf{x})\right)^2\right]}_{\textit{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\epsilon^2\right]}_{\textit{noise}}$$

Main Idea of Bagging

Bagging: average many models sensitive* training dataset to reduce variance.

* Related to overfitting.

Ensemble learning reduces variance by M times for independent models

Assuming $f_m(x) = z_m$ for $m = 1, \ldots, B$. Variance of the two random variables z_1 and z_2 is:

$$\begin{aligned}
\mathrm{Var}[z_1 + z_2] &= \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2 \\
&= \mathbb{E}[z_1^2 + z_2^2 + 2z_1 z_2] - (\mathbb{E}[z_1] + \mathbb{E}[z_2])^2 \\
&= \mathbb{E}[z_1^2] + \mathbb{E}[z_2^2] + \mathbb{E}[2z_1 z_2] - \mathbb{E}[z_1]^2 - \mathbb{E}[z_2]^2 - 2\mathbb{E}[z_1]\mathbb{E}[z_2] \\
&= (\mathbb{E}[z_1^2] - \mathbb{E}[z_1]^2) + (\mathbb{E}[z_2^2] - \mathbb{E}[z_2]^2) + 2(\mathbb{E}[z_1 z_2] - \mathbb{E}[z_1]\mathbb{E}[z_2]) \\
&= \mathrm{Var}[z_1] + \mathrm{Var}[z_2] + 2\underbrace{\mathrm{Cov}[z_1, z_2]}_{0 \text{ if } z_1 \perp\!\!\!\perp z_2}
\end{aligned}$$

Variance of the sum of B independent variables (the last equality assumes equal variance σ^2):

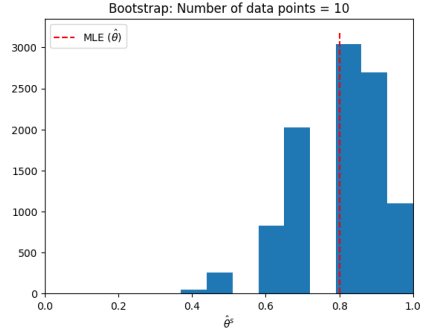
$$\mathrm{Var}[\sum_{m=1}^{M} z_m] = \sum_{m=1}^{M} \mathrm{Var}[z_m] \triangleq M\sigma^2$$

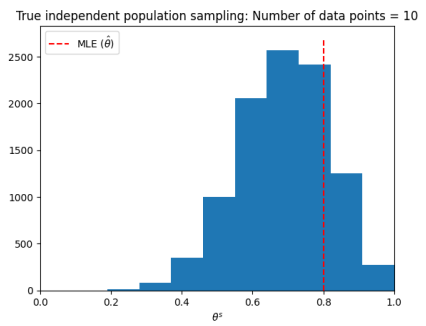
Variance of the average over M independent variables is M times smaller than their individual variances:

$$\mathrm{Var}[\frac{1}{M}\sum_{m=1}^{M} z_m] = \frac{1}{M^2}\mathrm{Var}[\sum_{m=1}^{M} z_m] = \frac{1}{M^2} M\sigma^2 = \frac{1}{M}\sigma^2$$

Bootstrap sampling

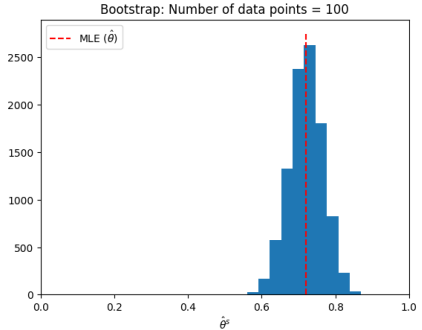
1. Suppose we have a single dataset $\mathcal{D}$ of size N (e.g., N coin tosses).
2. Sample *with replacement* to create M sets of datasets $\mathcal{D}^{(m)}$ each of size N.
3. For each dataset $\mathcal{D}^{(m)}$, train a model to obtain model parameter $\theta^{(m)}$ (e.g., the MLE of the Bernoulli rate from N coin tosses in $\mathcal{D}^{(m)}$ is $\theta^{(m)} = \frac{1}{N}\sum_{n=1}^{N} y_n^{(m)}$).
4. Average over the model parameters $\theta^{(m)}$ to approximate $\mathbb{E}_{\mathcal{D}}[\theta]$.
5. Histograms based on $B = 10K$ bootstraps each with N=10 samples (true $\theta = 0.7$). Notebook: https://www.cs.mcgill.ca/~isabeau/COMP551/W24/bootstrap_demo_bernoulli.ipynb.

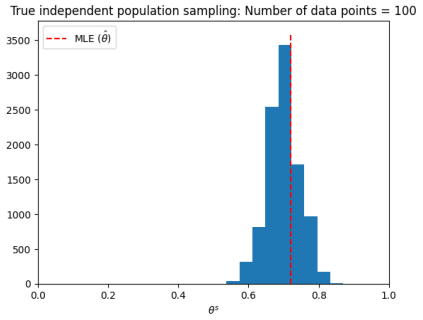




Bootstrap sampling

1. Suppose we have a single dataset $\mathcal{D}$ of size N (e.g., N coin tosses).
2. Sample *with replacement* to create M sets of datasets $\mathcal{D}^{(m)}$ each of size N.
3. For each dataset $\mathcal{D}^{(m)}$, train a model to obtain model parameter $\theta^{(m)}$ (e.g., the MLE of the Bernoulli rate from N coin tosses in $\mathcal{D}^{(m)}$ is $\theta^{(m)} = \frac{1}{N}\sum_{n=1}^{N} y_n^{(m)}$).
4. Average over the model parameters $\theta^{(m)}$ to approximate $\mathbb{E}_{\mathcal{D}}[\theta]$.
5. Histograms based on $B = 10K$ bootstraps each with N=100 samples (true $\theta = 0.7$). Notebook: https://www.cs.mcgill.ca/~isabeau/COMP551/W24/bootstrap_demo_bernoulli.ipynb.

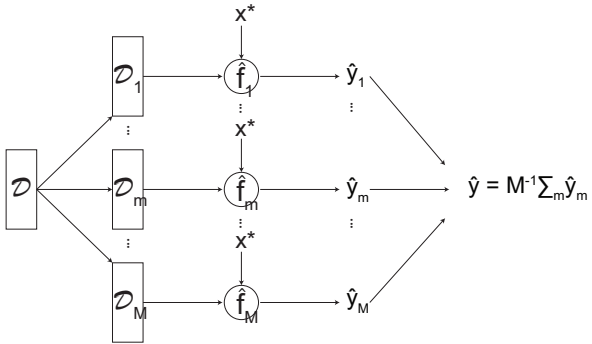




Bagging: Bootstrap aggregation (L. Breiman. Bagging predictors. In: *Machine Learning* **24** (1996))

Bagging training procedure is simple:

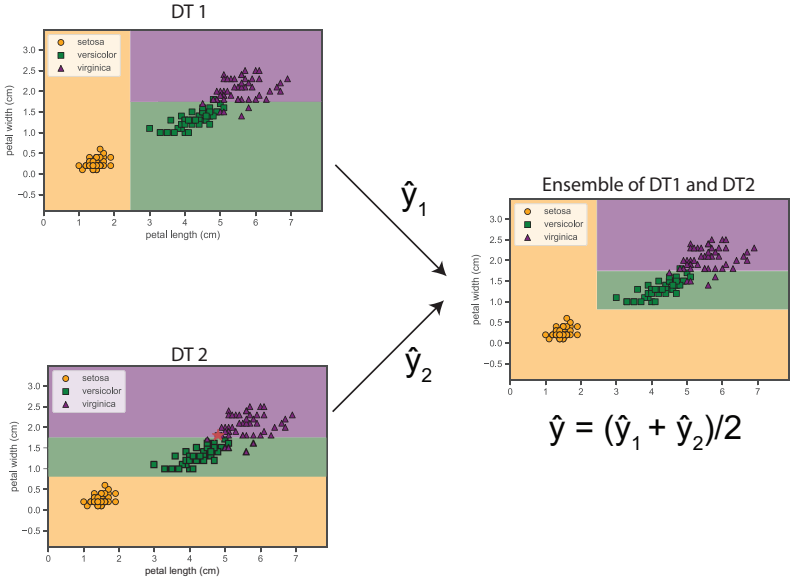
1. Suppose we have a labelled dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_n^{N=1}$.
2. Sample *with replacement* from $\mathcal{D}$ to get M datasets $\mathcal{D}^{(m)} = \{\mathbf{x}_n^{(m)}, y_n^{(m)}\}_n^{N=1}$.
3. Train M **base models** on the M datasets.
4. During testing, average the model predictions: $\hat{y} = \frac{1}{M}\sum_{m=1}^{M}\hat{y}_m$.
5. Bagging also estimates of model uncertainty: $\Delta\hat{y} = \left(\frac{1}{M}\sum_{m=1}^{M}(\hat{y}_m - \hat{y})^2\right)^{\frac{1}{2}}$.



Properties of ensemble models trained by the bagging algorithm

- Each model is trained on average, 63% of the *unique* examples:
  - The chance that a single data point will *not* be selected from the set of size N in any of the N draws is $(1 - 1/N)^N$
  - In the limit for large N, $\lim_{N\to\infty}(1 - 1/N)^N \to \exp(-1) \approx 0.37$
- The 37% of the training instances that are not used by a given base model are called **out-of-bag** (OOB) instances, which can be used as validation set for the base model (e.g., choosing tree depth).
- Bagging prevents the ensemble from relying too much on any individual training example, which improves robustness and generalization
- However, bagging does not always improve performance. The base model needs to be an **unstable estimator** (i.e. have high variance / be prone to overfitting) so that omitting some data changes the model fit.
- Also, the more independent the learners the better (pop quiz: why?).
- **Classification and regression trees** (CARTs) satisfy both conditions above but not other methods like KNN, SVM or linear regression.

Ensemble of two decision trees fit to the Iris flower dataset



Outline

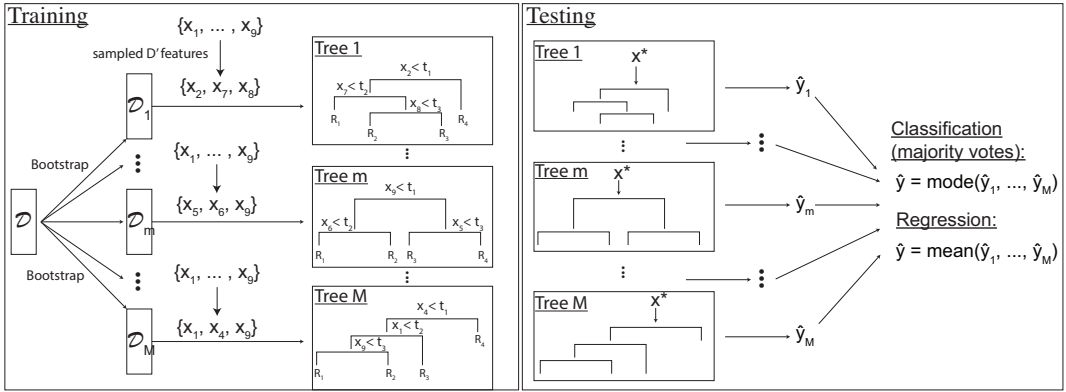
Main Idea of Random Forest

Random Forest: bagging of Decision Trees + restricting to a Random subset of Features* at each node.

* Some people call Random Features "Feature Bagging" even though it is unrelated to Bootstrapping.

Random Forest (RF) [L. Brieman *Machine Learning* 45.1 (2001), pp. 5-32]

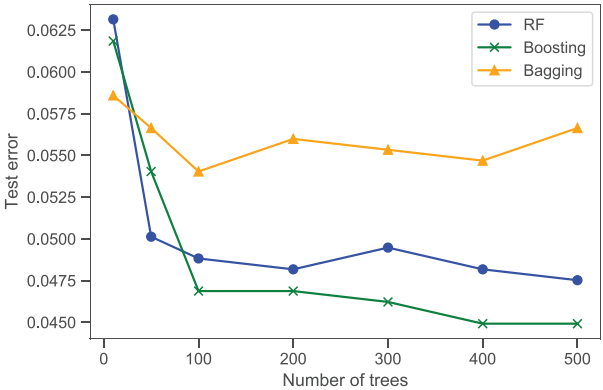
- Bagging learns diverse base models on subsets of bootstrap-sampled training examples. RF seeks to further *decorrelate* the base models (i.e., decision tree).
- An RF learns K decision or regression trees, where each tree is trained based on
  1. a bootstrap sample of N training data points and
  2. a randomly chosen subset of $\sqrt{D}$ (by default) input features at each tree node
- Out of bag (OOB) samples for each tree are used as validation to determine tree depth.



Spam detection using hand-crafted tabular data of 57 features

4601 email messages labelled with spam ($y = 1$) or non-spam ($y = 0$). The data was made available by George Forman from Hewlett-Packard. The 57 input features are:

- 48 keyword frequencies (e.g., `business`, `address`, `internet`, `George` (user name)),
- 6 special characters (`;` `.` `[` `$` `#`),
- 3 features corresponding to average, max, total length of capital letters.



RF performs much better than Bagging but worse than Boosting (discussed shortly).

Computing feature importance from RF

- Trees are popular because they are interpretable.
- Ensemble of trees such as RF lose that property
- To extract feature importance from a trained RF, we can follow these steps:
  1. Compute feature importance score for each feature d from a single decision tree classifier m by summing over all non-leaf nodes where feature d is used weighted by the reduction of cost (e.g., gini-index):

$$R_d(T_m) = \sum_{j=1}^{J} G_j \mathbb{I}[v_j = d]$$

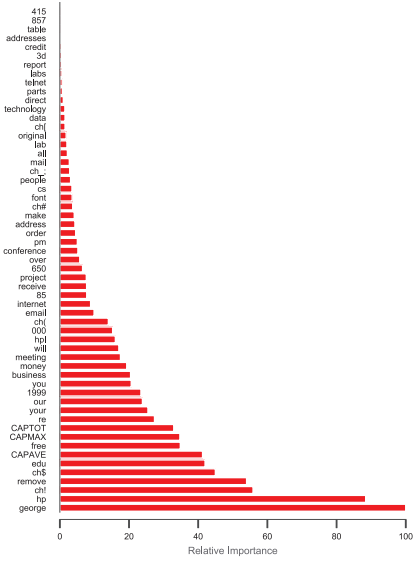
  2. Average over all trees in the RF:

$$R_d = \frac{1}{M} \sum_{m=1}^{M} R_d(T_m)$$

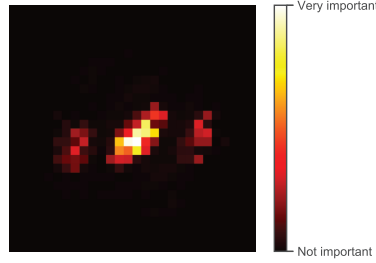
  3. Normalize the score: $\hat{R}_d = 100 \times R_d / \max(\mathbf{R})$ so that the most important score is 100%

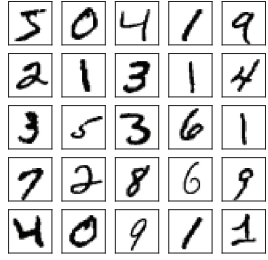
Feature importance scores for spam detection and MNIST classification

Feature importance of spam detection



Feature importance of MNIST classification





Ensemble Learning part II: Boosting

COMP 551 Applied Machine Learning

Isabeau Prémont-Schwarz
School of Computer Science
McGill University

Fall 2024

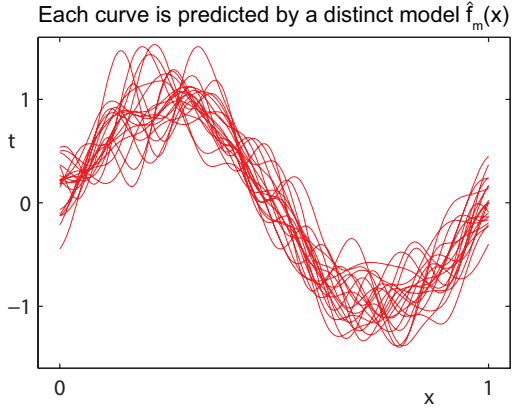
Please complete the online evaluation for the course on Mercury.

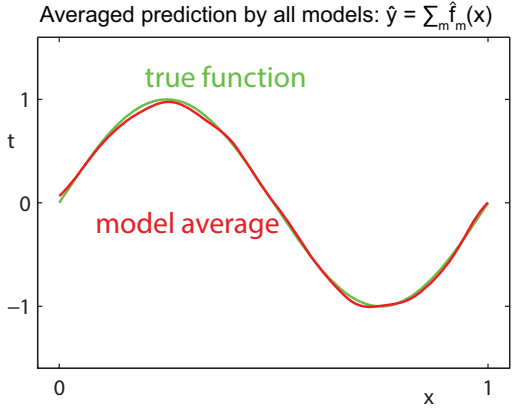
https://go.blueja.io/qLhY92bmaEW4w8hLQ53-LA

Your feedback is very important to us.

Thank you!

Recall Generalization Lectures & A2: model averaging reduces variance





$$\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}(\mathbf{x};\mathcal{D}) - y\right)^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right)^2\right]}_{\textit{Variance}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(\mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(\mathbf{x})] - f(\mathbf{x})\right)^2\right]}_{\textit{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\epsilon^2\right]}_{\textit{noise}}$$

Outline

Main Idea of Boosting

Boosting: sequentially adding many weak models to create a powerful model.

(pop quiz: linear models, trees, single feature linear models?)

Boosting is a sequential fitting algorithm

Model f_m is a sum of weak learners F_i: $f_m(\mathbf{x}) = \sum_i \beta_i F_i(\mathbf{x})$, where $\beta_i \in \mathbb{R}^+$.

- Regression tasks: $\hat{y} = f_m(\mathbf{x})$
- Binary classification tasks $\hat{y} = p(y = 1|\mathbf{x}) = \sigma(f_m(\mathbf{x})) = \frac{1}{1+\exp(-f_m(\mathbf{x}))}$.
- Multi-class classification: (pop quiz: ?)

1. Fit function F_1 on the original training data
2. Fit model F_m's parameters θ_m to solve the residual errors of f_{m-1}.
3. Find the weight β_m of F_m which will minimize the original loss.
4. Repeat 2 and 3 until we have the desired M models $F_1, \ldots, F_M$.
5. The final ensemble model prediction is

$$f(\mathbf{x}, \Theta) = \sum_m \beta_{m=1}^{M} F_m(\mathbf{x}; \theta_m)$$

Forward stagewise additive modelling

Goal: minimize the prediction loss on N training data points w.r.t. the function:

$$\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$$

Strategy: sequentially minimize the loss at each iteration m:

1. Obtain optimal parameters for the base model function $F(\mathbf{x}, \theta)$ (**weak learner**) and its weight:

$$\beta_m, \theta_m \leftarrow \arg\min_{\beta,\theta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F(\mathbf{x}^{(n)}; \theta))$$

2. Set the ensemble function (**strong learner**) at the iteration m to be:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$$

Note: the detailed boosting algorithm depends on the loss function ℓ. The most intuitive loss function in this context is the *quadratic loss* and the fitting algorithm is called **least squares boosting** (L2Boosting) (next).

Quadratic loss and least squares boosting (L2Boosting)*

Squared error loss:

$$\begin{aligned}\ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + F(\mathbf{x}^{(n)}; \theta)) &\equiv (y^{(n)} - f_{m-1}(\mathbf{x}^{(n)}) - F(\mathbf{x}^{(n)}; \theta))^2 \\ &\equiv (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2\end{aligned}$$

- $r_m^{(n)} = y^{(n)} - f_{m-1}(\mathbf{x}^{(n)})$ is the current residual error on the n'th training example.
- Therefore, $F(\mathbf{x}^{(n)}; \theta)$ further reduces the residual error made by $f_{m-1}(\mathbf{x}^{(n)})$.
- Overall fitting algorithm of least squares boosting then becomes:

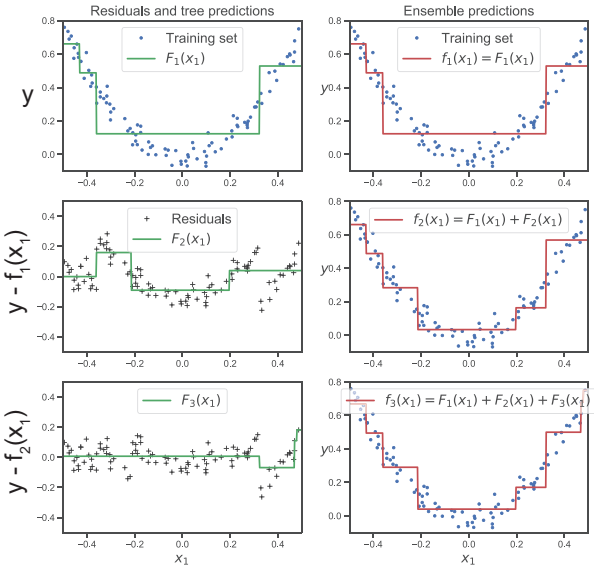
$$\theta_m \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

$$f_m(\mathbf{x}^{(n)}) = f_{m-1}(\mathbf{x}^{(n)}) + F(\mathbf{x}^{(n)}; \theta_m)$$

- Note: the function $F(\mathbf{x}^{(n)}; \theta)$ does not need to be differentiable. For example, we can use regression tree as the base model.

* For simplicity we give here the algorithm using fixed $\beta = 1$ because in L2Boost the β's are redundant if the parameters of θ of F can scale F by a multiplicative constant as is almost always the case.

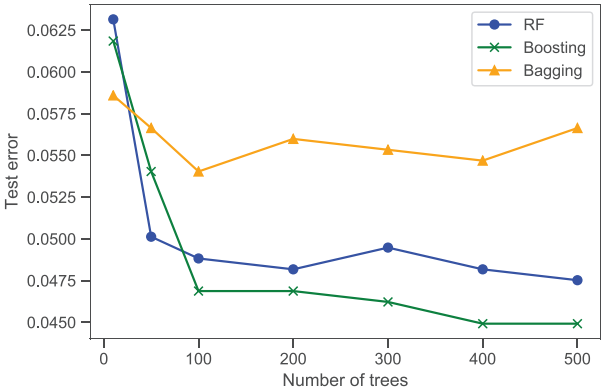
Least square boosting using regression trees when β fixed to 1



Spam detection using hand-crafted tabular data of 57 features

4601 email messages labelled with spam ($y = 1$) or non-spam ($y = 0$). The data was made available by George Forman from Hewlett-Packard. The 57 input features are:

- 48 keyword frequencies (e.g., `business`, `address`, `internet`, `George` (user name)),
- 6 special characters (`;` `.` `[` `$` `#`),
- 3 features corresponding to average, max, total length of capital letters.



RF performs much better than Bagging but worse than Boosting.

Gradient Boosting applied to different ℓ's gives most Boosting Algos

Name	ℓ	y	f
L2Boost	$\frac{1}{2}(y-\hat{y})^2$	$\mathbb{R}$	$\hat{y} = f(\mathbf{x})$
AdaBoost	$\exp(-yf) = \left(\frac{1-\hat{y}}{\hat{y}}\right)^{\frac{y}{2}}$	$\{-1, 1\}$	$\hat{y} = p(y=1) = \sigma(2f(\mathbf{x}))$
LogitBoost	$-(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$	$\{0, 1\}$	$\hat{y} = p(y=1) = \sigma(f(\mathbf{x}))$

LogitBoost example:

$$
\begin{aligned}
\ell(y, f) &= -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})) \\
&= -(y\log(\frac{1}{1+\exp(-f(x))}) + (1-y)\log(1-\frac{1}{1+\exp(-f(x))})) \\
&= (y\log(1+\exp(-f(x))) + (1-y)\log(1+\exp(f(x))))
\end{aligned}
$$

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting: The General Case

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\Longrightarrow$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*initialization* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \ell(y^{(n)}, F(\mathbf{x}^{(n)}; \theta))$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = - \left. \frac{\partial \ell(y^{(n)}, f)}{\partial f} \right|_{f = f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate functional gradients/residuals*): $r_m^{(n)} = - \left. \frac{\partial \ell(y^{(n)}, f)}{\partial f} \right|_{f = f_m(\mathbf{x}^{(n)})}$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})} = -(f_m(\mathbf{x}^{(n)}) - y^{(n)})$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})} = y^{(n)} - f_m(\mathbf{x}^{(n)})$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})} = y^{(n)} - f_m(\mathbf{x}^{(n)})$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})} = y^{(n)} - f_m(\mathbf{x}^{(n)})$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})} = y^{(n)} - f_m(\mathbf{x}^{(n)})$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Gradient Boosting + L2 loss $\implies$ L2Boosting

Goal: minimize $\mathcal{L}(f) = \sum_{n=1}^{N} \ell(y^{(n)}, f(\mathbf{x}^{(n)}))$

Algorithm: sequentially minimize the loss at each iteration m:

- **Step 1**(*init.* $m = 1$): Fit $f_1 = F_1$ by $\theta_1 \leftarrow \arg\min_{\theta} \sum_{n=1}^{N} \frac{1}{2} \left(y^{(n)} - F(\mathbf{x}^{(n)}; \theta)\right)^2$
- **Step 2**(*calculate gradients/residuals*): $r_m^{(n)} = -\left.\frac{\partial \ell(y^{(n)}, f)}{\partial f}\right|_{f=f_m(\mathbf{x}^{(n)})} = y^{(n)} - f_m(\mathbf{x}^{(n)})$
- **Step 3**(*fit* F_m *to residual*):

$$\theta_m \leftarrow \arg\min_{\theta} \sum_{i=1}^{N} (r_m^{(n)} - F(\mathbf{x}^{(n)}; \theta))^2$$

- **Step 4**(*line search* β_m):

$$\beta_m \leftarrow \arg\min_{\beta} \sum_{n=1}^{N} \ell(y^{(n)}, f_{m-1}(\mathbf{x}^{(n)}) + \beta F_m(\mathbf{x}^{(n)}; \theta_m))$$

- **Step 5**(*update* f): $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m F(\mathbf{x}; \theta_m)$
- **Step 6**(*iterate*): $m+ = 1$, goto step 2.

Extreme Gradient Boosting (XGBoost)

- F's are Decision Trees.
- 2nd order Taylor expansion (as opposed to 1st order only for Gradient Boosting).
- Adds regularization terms.

Boosting versus Bagging/RF

- While bagging and RF reduce the variance by fitting independent trees, boosting reduces the bias by *sequentially* fitting classifiers that depend on each other.
- While boosting is slower than bagging and RF because of its sequential fitting algorithm, in practice it often produces better performance as we saw in the earlier spam detection application.
- In Bagging $\beta_m = 1/M$ but in Boosting the β_m's are fitted.