Statistical Machine Learning
Ensemble Methods and Random Forest

Horacio Gómez-Acevedo
Department of Biomedical Informatics
University of Arkansas for Medical Sciences

February 28, 2023

UAMS

No Free Lunch Theorem

To the general questions

- If we are interested solely in the generalization performance, are there any reasons to prefer one classifier or learning algorithm over another?
- If we make no prior assumptions about the nature of the classification task, can we expect any classification method to be superior or inferior overall?
- Can we even find an algorithm that is overall superior to random guessing?

The answer to these questions is NO. The so-called **No free lunch theorem** proves that there are no context-independent or usage-independent reasons to favor one learning or classification method over another.
The reference (Duda, Hart, and Stork, 2001) describes this theorem in more detail.

Ensemble Learning

We know that different learning algorithms can have different accuracies. There is no "ideal" one but we can combine them to attain better accuracy.
Ensemble learning techniques build a number of different predictors (base learners), then combine them to form the composite predictor to classify the test set. This phenomenon is known as *diversity*. Two classifiers are said to be diverse if they make different incorrect predictions on new data points.
Ensemble learning is often called a **committee machine**. And each of the classifiers composing the ensemble can be constructed either independently (e.g., bagging) or sequentially (boosting)

Bagging

We have seen that regression (or classification) trees are very useful but particularly unstable. That is, they suffer from high variance when presented with new data.
Fortunately, there is a general purpose procedures to reduce variance called *Bootstrap aggregation* or *bagging*.
Key Idea: Given a set of m independent (and identically distributed) observations $X_1, \ldots, X_m$ each with variance σ^2, the variance of the mean $\overline{X}$ is given by

$$\hat{\sigma}^2(\overline{X}) = \frac{1}{m}\sigma^2$$

Averaging a set of observations reduces variance

Bagging (cont)

It seems reasonable that for a given model $Y = f(X) + \varepsilon$ to get an estimate response at the point $X = x$ by averaging $\hat{f}^1(x), \ldots, \hat{f}^B(x)$ using B separate training sets, that is

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

where $\hat{f}^k$ is a tree regression estimate (without pruning) based on the observations $\{x_{k1}, \ldots, x_{kB}\}$.
Since we don't have multiple training sets, then we exploit **bootstrap!**. More precisely, for a bootstrapped training set b, we obtain the estimate $\hat{f}^{*b}(x)$, and the corresponding **bagging estimate**

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

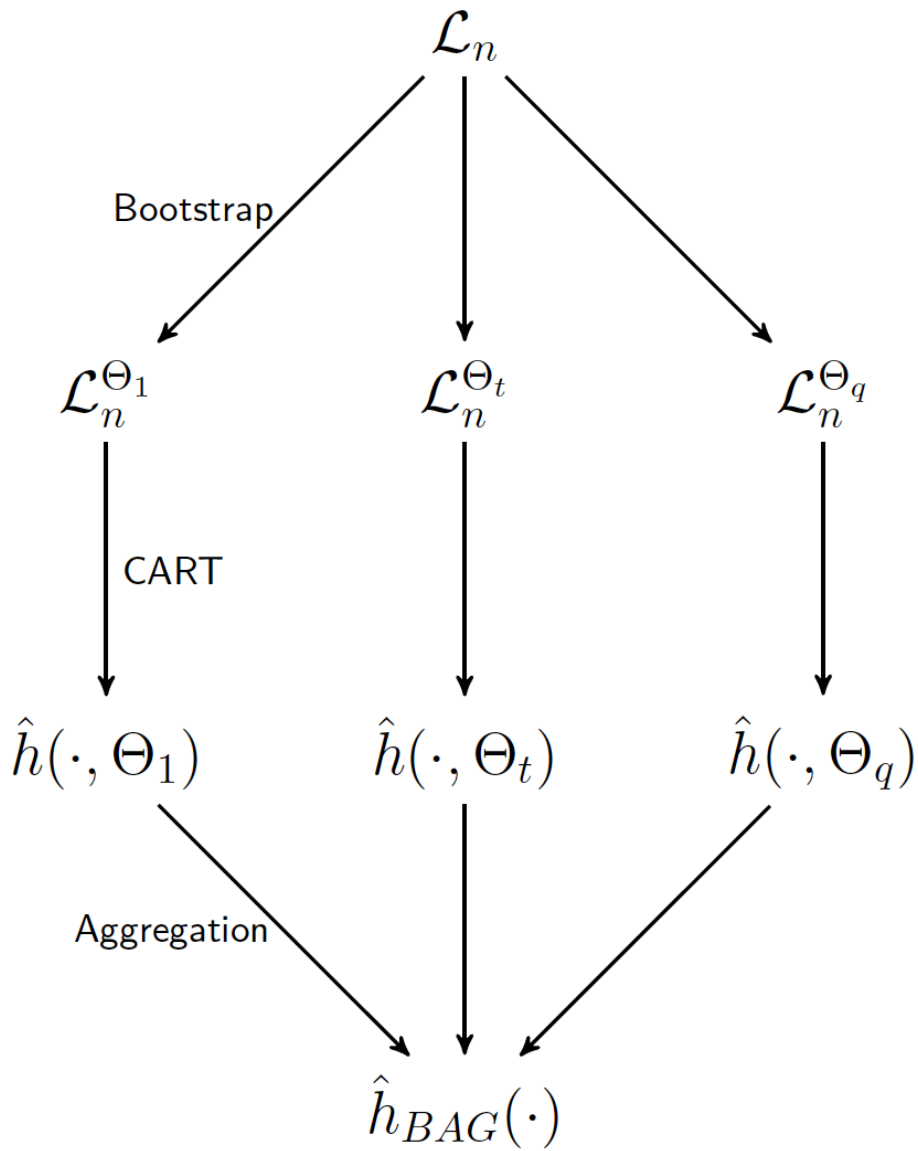
Bagging for Regression Trees

There are few steps that we need to take for regression trees

- Construct B regression trees using B bootstrapped training sets (without pruning or use CART)
- Average the resulting predictions

Keep in mind that each (bootstrapped) tree has low bias but high variance. The bagging procedure will reduce the variance at the expense of a "modest" increase in bias.

Bagging for Regression/Classification Trees



Bagging for Classification Trees

The steps are similar for classification trees

- Construct B regression trees using B bootstrapped training sets (without pruning)
- The prediction will be based on the majority vote. That is, the class most commonly occurring will be selected.

Out-of-Bag Observations

The main idea of the bootstrap is that from n observations, we select a sample with replacement m observations.
What is the probability of **not** selecting sample 1 ?
The probability of picking sample different from 1 would be $(1 - \frac{1}{m})$. Since we are repeating the experiment with replacement, the probability that a bootstrap sample does not contain sample 1 is

$$\left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{1}{m}\right) \cdots \left(1 - \frac{1}{m}\right) = \left(1 - \frac{1}{m}\right)^m$$

A little bit of calculus shows that

$$\lim_{m \to \infty} \left(1 - \frac{1}{m}\right)^m = \exp(-1) \approx 36.79\%$$

Thus, bootstrapping will not touch about $1/3$ of the observations! and those observations are referred to as **Out-of-Bag (OOB)**.

OOB Error Estimation

We can exploit the OOB observations to estimate the test error in the bagging process without the need of cross-validation or even a split of the data in training and testing.
Once we have obtained our $\hat{f}_{bag}(x)$, we can use the OOB observations (i.e., observations not used for the bagging estimation) to determine predictions.
More precisely, we obtain $\hat{f}^i_{\text{oob}}(x)$ based on the OOB observations $\{x_{i1}, \ldots, x_{iK}\}$, where $K \approx B/3$ for B big enough.

$$\hat{f}_{OOB}(x) = \frac{1}{K} \sum_{i=1}^{K} \hat{f}^i_{\text{oob}}(x)$$

This procedure leads to the calculation of the test MSE that is a valid estimate since the response is derived from trees that were not involved in the bagged model.
A similar expression is valid for classification, but instead of the average we can use the majority vote and purity metrics instead of MSE or RSS.

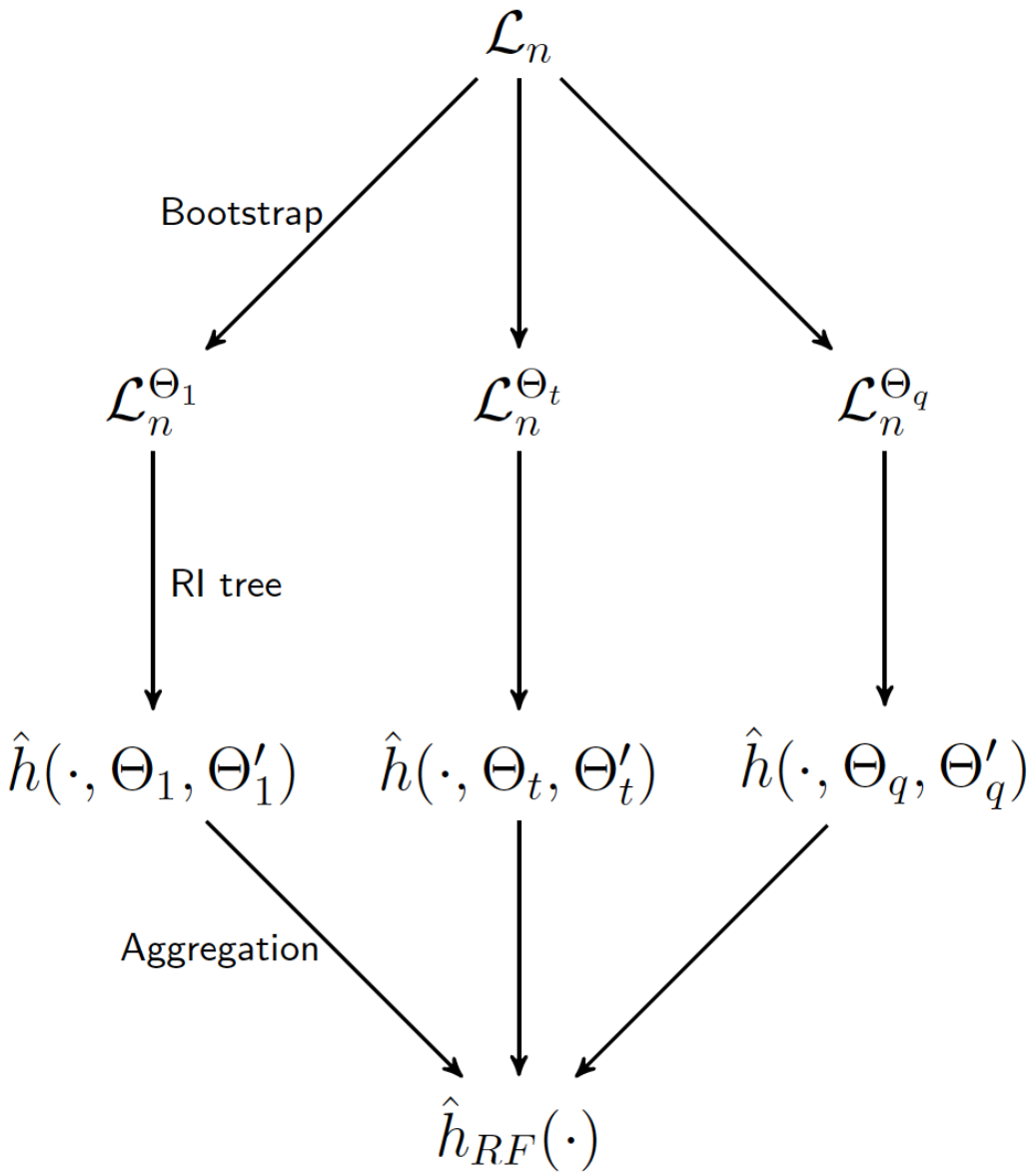
Notes on Bagging

- Bagging tries to average many noisy but approximately unbiased models, hence reducing the prediction variance without affecting the prediction bias.
- It works best for high-variance, low-bias procedures.
- Bias in bagged trees is the same as that of the individual trees.

Random Forest (with Random Inputs)

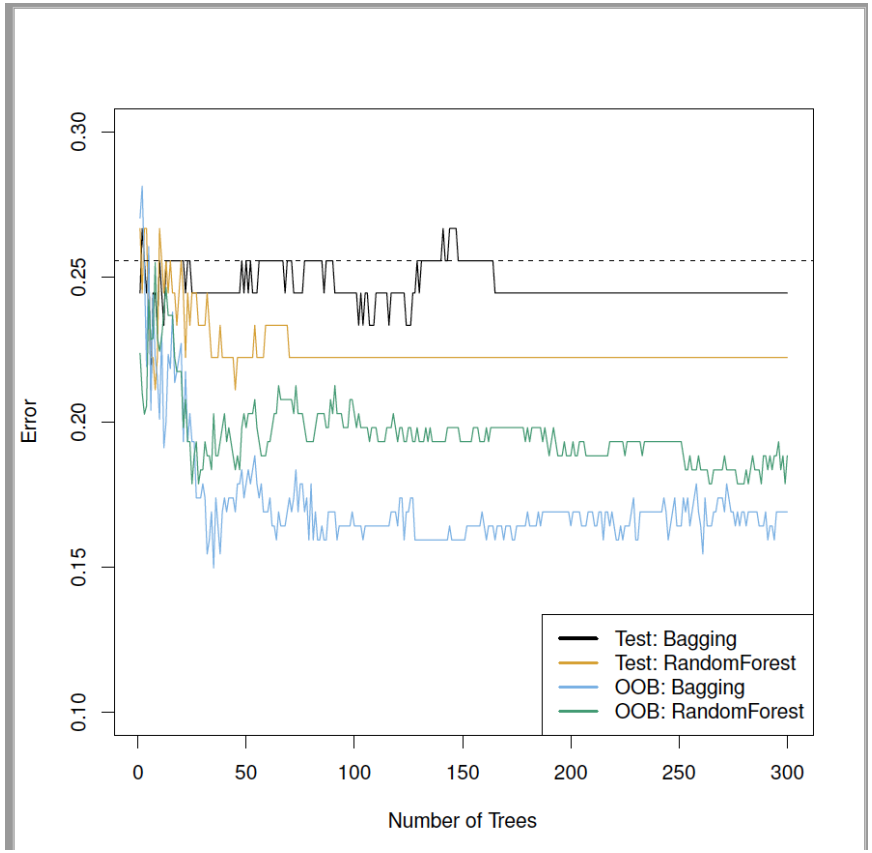
It follows similar rationale as in bagging but with an interesting random twist.
We build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a *random sample of m predictors* is chosen as split candidates from the full set of p predictors. We normally set $m \approx \sqrt{p}$.
What is the advantage of random forest over bagging?
When we have a strong predictor, bagging trees will consider that predictor frequently, thus bagging trees will look alike. By having a random choice on the predictors, we may generate "different" trees that otherwise we would not have explored. This process is referred to as *decorrelating trees*.

Random Forest (with Random Inputs)



Random Forest vs Bagging

The test errors from the `Heart` data are depicted below

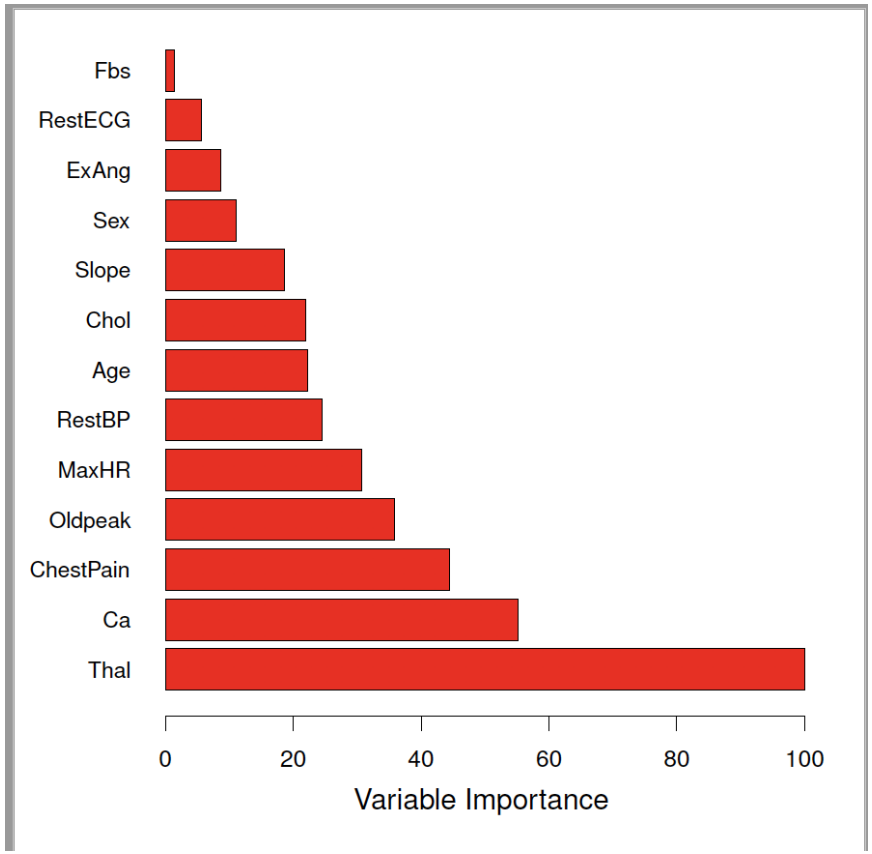


Variable Importance Measures

We know that bagging improves the accuracy in our predictions, at the expense of making our models harder to interpret.
For bagging regression trees, we use the **Variable Importance Measure (VIM)** that is defined as the total amount that the RSS is decreased due to splits over the given predictor, averaged over all B trees. The larger the VIM, the more "relevant" is that predictor.
For bagging classification trees, we can define VIM as the total amount that the Gini index (or cross-entropy) is decreased by splits over a given predictor, averaged over all B trees.

VIM example

The `Heart` data set VIM plot with a mean decrease of Gini index and normalized VIM is shown below.



Variable of Importance (Details)

One implementation of the variable of importance based on OOB calculation is as follows:

- Suppose you have $\{X_1, \ldots, X_p\}$ variables and fix X_j.
- Consider the bootstrap of the sample $\mathcal{L}_n^{\Theta_s}$ and the associated OOB_s sample (i.e., all the observations that do not belong to $\mathcal{L}_n^{\Theta_s}$)
- Calculate errOOB_s the error made on OOB_s by the tree build on $\mathcal{L}_n^{\Theta_s}$. For instance MSE or misclassification rate.
- Randomly permute the values of the variable X_j in the OOB_s sample. This gives a permuted samples, noted as $\widetilde{OOB}_s^j$.
- Calculate the err$\widetilde{OOB}_s^j$, the error made on $\widetilde{OOB}_s^j$ by the tree build on $\mathcal{L}_n^{\Theta_s}$.
- Repeat these operations for all bootstrap samples.

Variable of Importance (Details cont.)

$$VI(X_j) = \frac{1}{q}\sum_{s=1}^{q}(\text{err}\widetilde{OOB}_s^j - \text{err}OOB_s)$$

Thus, the error increase originating from the random permutations of the variable suggests the importance of such variable.
Note that there are several representations of the VI expression (at least in R). For instance, we can have the normalized version, but there are other methodologies based on the Mean Decreased impurity (MDI). Just verify what method are you plotting and what is the valid range.

Boosting

Boosting or ARCing (adaptive resampling and combining) is another general methodology to improve the predictions from a decision tree.
Boosting algorithms belong to a class of voting methods that produce a classifier as a linear combination of weak classifiers.
For tree-based methods, boosting makes trees grown sequentially as they gather information from previously generated trees.
Boosting does not require bootstrap sampling as each tree is fit on a modified version of the original data set.

Boosting Idea

We will describe a method referred to as AdaBoost.M1. Consider the 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 taken one of the two values $\{-1, 1\}$. The error rate on the training samples is

$$\overline{\text{err}} = \frac{1}{N} \sum_{j=1}^{N} I(y_j \neq G(x_j)) \quad , I(x) \text{ is the indicator function}$$

Boosting Idea (cont)

The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified version of the data, thereby producing a sequence of weak classifiers $G_m(x)$, $m = 1, \ldots, M$. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction:

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

The α_j are computed by the boosting algorithm and weight the contribution of each respective $G_m(x)$. Their effect is to give higher influence to the more accurate classifiers in the sequence.

Adaboost

The data modification at each boosting step consists of applying weights $w_1, \ldots, w_N$ to each of the training observations (x_i, y_i), $i = 1, \ldots, N$. Initially all of the weights are set to $w_i = 1/N$, so that the first step simply trains the classifier on the data in the usual manner. For each successive iteration $m = 2, 3, \ldots, M$ the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. At step m, those observations that were misclassified by the classifier $G_{m-1}(x)$ induced at the previous step have their weights increase, whereas the weights are decreased for those that were classified correctly. Thus, as iterations proceed, observations that are difficult to correctly classify receive ever-increasing influence. Each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence.

Adaboost (Algorithm)

- Initialize the observation weights $w_i = 1/N$ for $i = 1, \ldots, N$.
- For $m = 1$ to M do:
  1. Fit a classifier $G_m(x)$ to the training data using weights w_i.
  2. Compute
  $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$
  3. Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$
  4. $w_i \leftarrow w_i \exp(\alpha_m \cdot I(y_i \neq G_m(xi))$ for $i = 1, \ldots, N$.
- Output $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$

Boosting for Regression Trees

Given a current model, we fit a decision tree to the residuals from the model rather than the outcome Y as the response. And we add these residuals to a new decision tree and update again the residuals.
Boosting has the following tuning parameters:

- B that represents the number of trees. Do not take very large values of B as boosting tends to over-fit. We can use cross-validation to determine a good candidate for B.
- The shrinkage parameter λ controls the learning rate.
- The number of splits d controls the complexity of the boosted ensemble. Sometimes $d = 1$ works well.

Boosting Algorithm for Regression Trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. for $b = 1, \ldots, B$ repeat:
   2.1 Fit a tree $\hat{f}^b$ with d splits to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   2.3 Update the residuals,

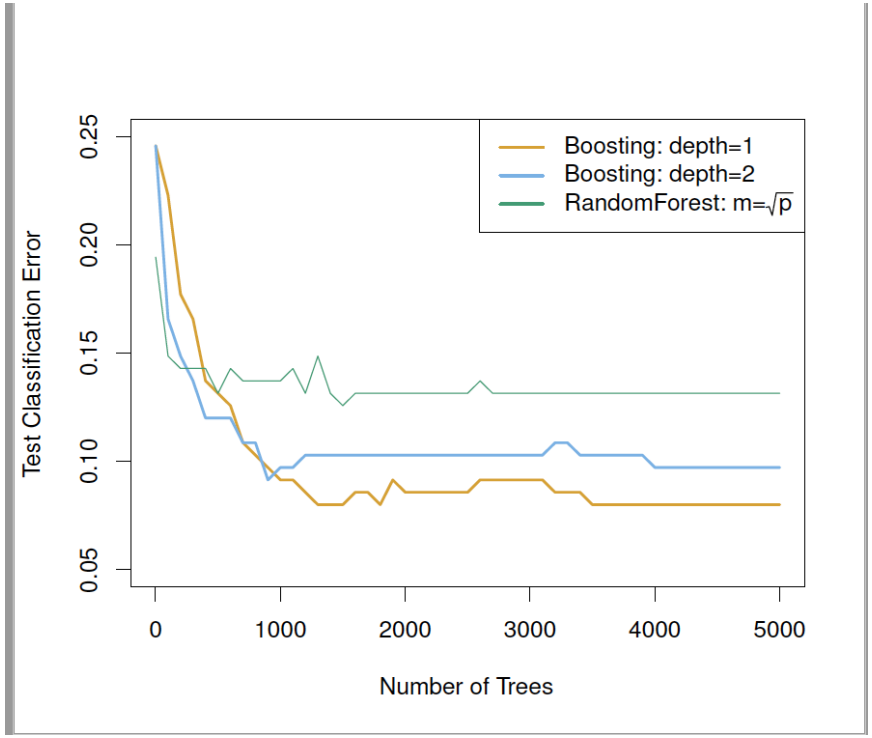
$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

3. Output the boosted model

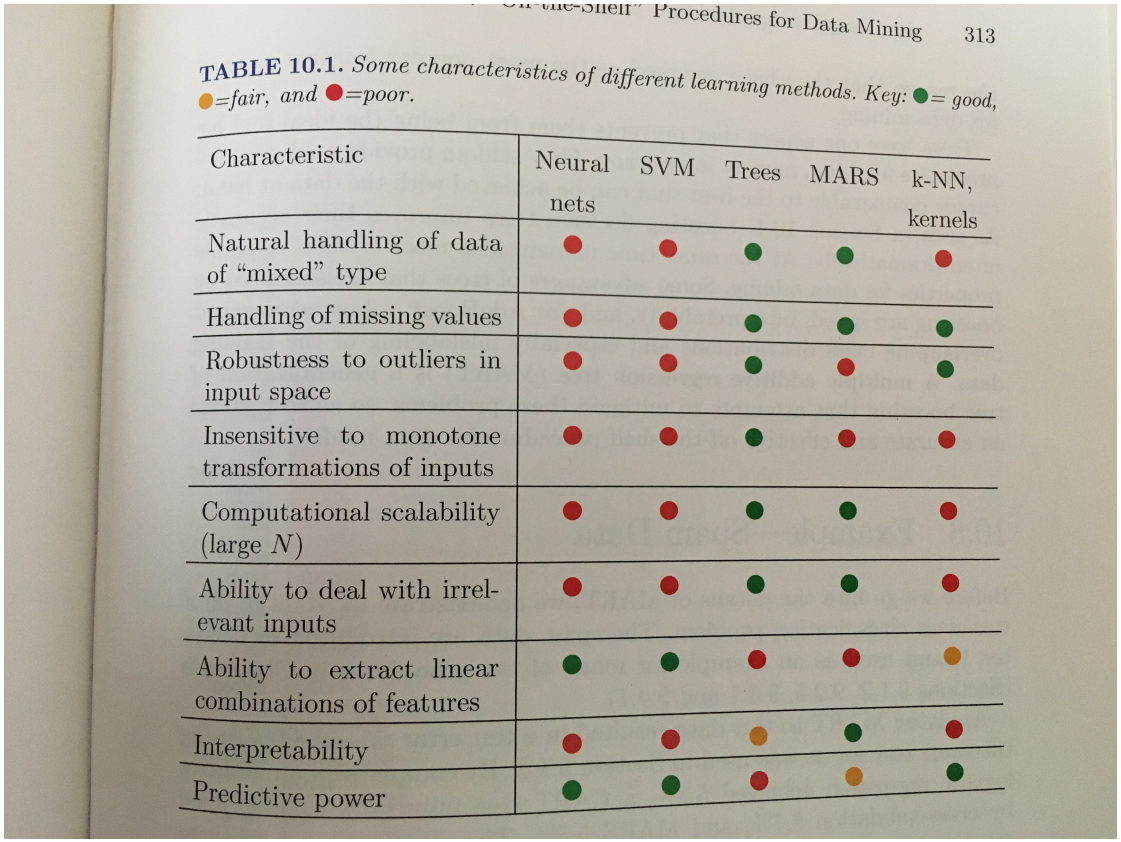
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x) \tag{1}$$

Boosting

Boosting and random forest comparison in a 15-class gene expression data set to predict cancer.



Final Thoughts



...the-Shelf" Procedures for Data Mining 313

TABLE 10.1. *Some characteristics of different learning methods. Key:* ● *= good,* ● *=fair, and* ● *=poor.*

Characteristic	Neural nets	SVM	Trees	MARS	k-NN, kernels
Natural handling of data of "mixed" type	●	●	●	●	●
Handling of missing values	●	●	●	●	●
Robustness to outliers in input space	●	●	●	●	●
Insensitive to monotone transformations of inputs	●	●	●	●	●
Computational scalability (large N)	●	●	●	●	●
Ability to deal with irrelevant inputs	●	●	●	●	●
Ability to extract linear combinations of features	●	●	●	●	●
Interpretability	●	●	●	●	●
Predictive power	●	●	●	●	●

References

Materials and some of the pictures are from (Gareth et al., 2015), (Genuer and Poggi, 2020), (Du and Swamy, 2019), and (Hastie, Tibshirani, and Friedman, 2001).

- Du, K-L. and N. S. Swamy (2019). *Neural Networks and Statistical Learning*. Second Edition. Springer. ISBN: 978-1-4471-7451-6.
- Duda, R. O., P. E. Hart, and D.G. Stork (2001). *Pattern Classification*. 2nd. Edition. John Wiley & Sons. ISBN: 978-0-471-05669-0.
- Gareth, J. et al. (2015). *An Introduction to Statistical Learning*. 1st edition. Springer. ISBN: 978-1-4614-7137-0.
- Genuer, R. and Jean-Michel Poggi (2020). *Random Forests with R*. Use R! ISBN: 978-3-030-56484-1.
- Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Elements of Statistical Learning*. 1st Edition. Springer Series in Statistics. Springer. ISBN: 978-0-387-95284-0.

I have used some of the graphs by hacking TiKz code from StakExchange, Inkscape for more aesthetic plots and other old