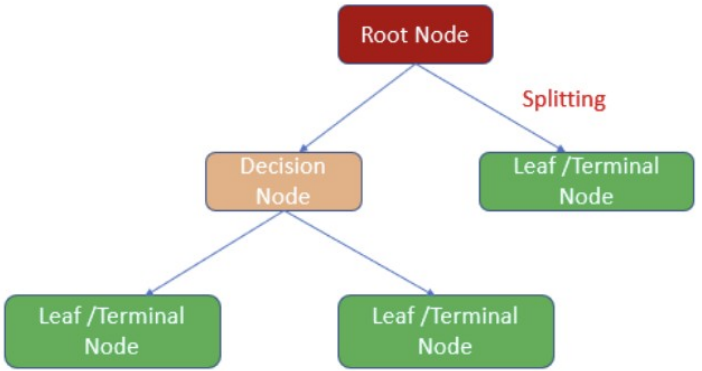
Decision Trees



The algorithm of the decision tree models works by repeatedly partitioning the data into multiple sub-spaces so that the outcomes in each final sub-space is as homogeneous as possible. This approach is technically called *recursive partitioning*. The produced result consists of a set of rules used for predicting the outcome variable, which can be either:

- a continuous variable, for regression trees
- a categorical variable, for classification trees

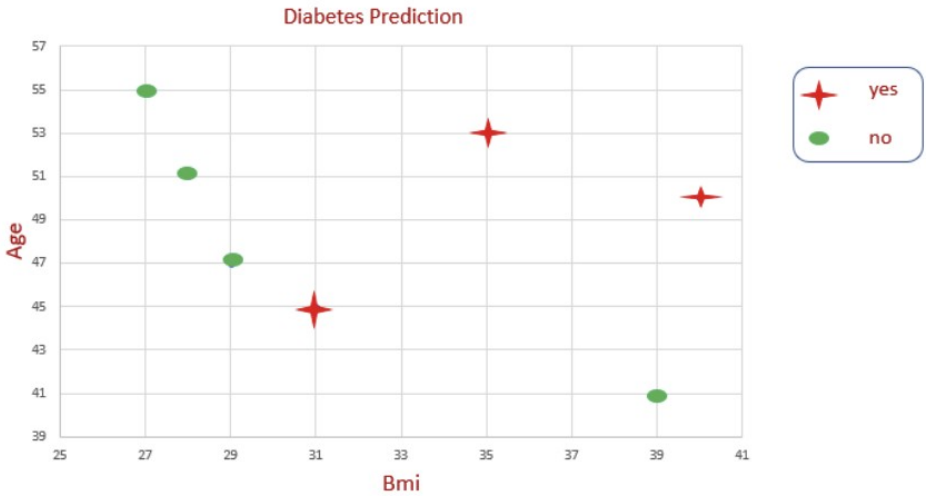
Root Node → The topmost node in the decision tree is known as the *root node*.
Decision Node → The subnode which splits into further subnodes is known as the *decision node*.
Leaf/Terminal Node → A node that does not split is known as a *leaf node/terminal node*.

Classification Trees

Data Set

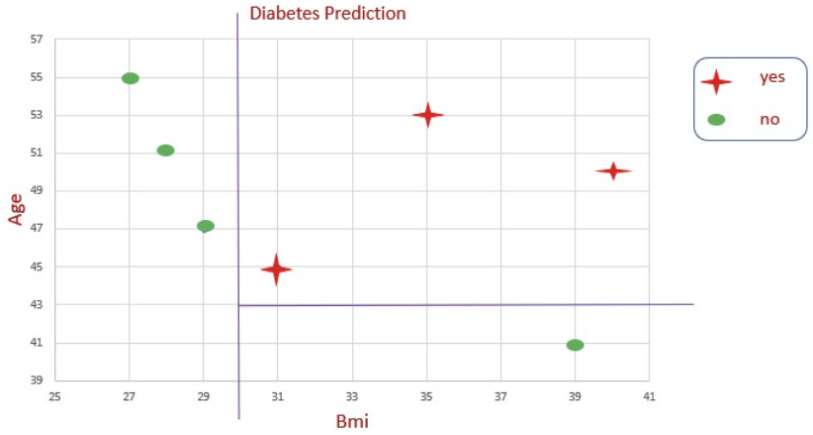
Bmi	Age	Diabetes
31	45	yes
29	47	no
27	55	no
35	53	yes
28	51	no
40	50	yes
39	41	no

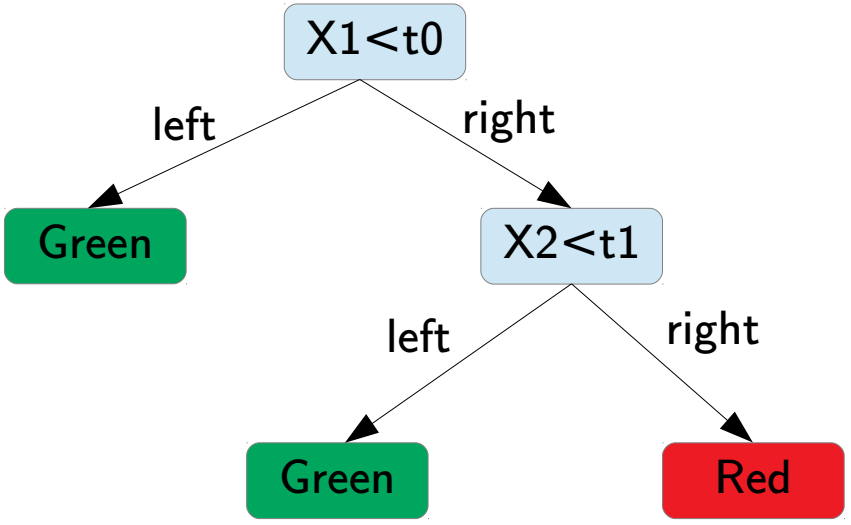
Data Set Representation



Let's predict if a person of a given age and BMI will have diabetes or not.

Decision tree algorithm





We can't draw a single line to get a decision boundary. We are splitting the data again and again to get the decision boundary. This is how the decision tree algorithm works.

Measures Used for Split

- **The process to calculate Gini Measure:**

$$Gini = 1 - \sum_j p_j^2$$

where **P(j)** is the Probability of Class j

- **Entropy**: Entropy is a way to measure impurity.

$$Entropy = -\sum_j p_j \log_2 p_j$$

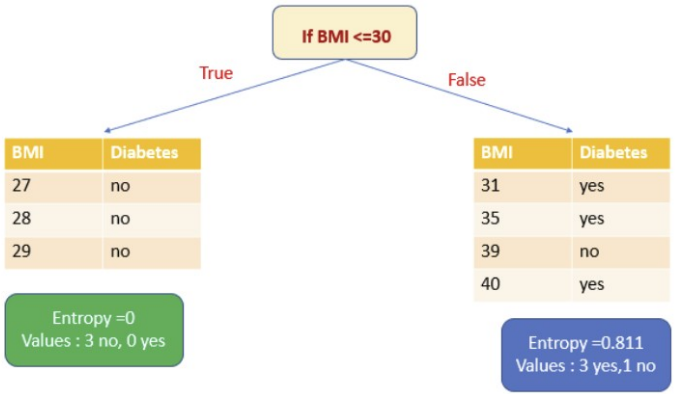
- The **Information Gain (IG)** can be defined as follows:

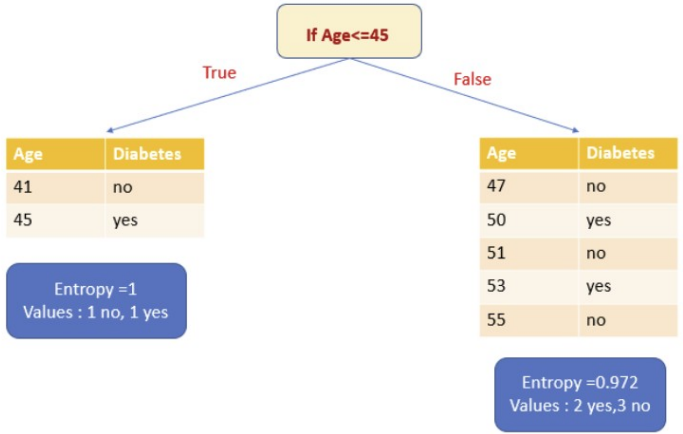
$$IG(D_p) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

Where *I* could be **entropy or Gini index, D** (p), D(Left), and D(Right) are the dataset of the parent, left, and right child node.

How does splitting occur?

Bmi	Age	Diabetes
31	45	yes
29	47	no
27	55	no
35	53	yes
28	51	no
40	50	yes
39	41	no





Entropy= $-(p(0) * \log(P(0)) + p(1) * \log(P(1)))$

$= -(4/7 * \log_2(4/7) + 3/7 * \log_2(3/7))$

$= -((-0.4617) + (-0.523))$

$= -(-0.985)$

Entropy=0.985

IG(BMI)=0.985-(3/7 *0)-(4/7*0.811)

= 0.985-0-0.463

IG(BMI) = 0.522

IG(Age) =0.985-(2/7 *1)-(5/7*0.972)

= 0.985-0.286-0.694

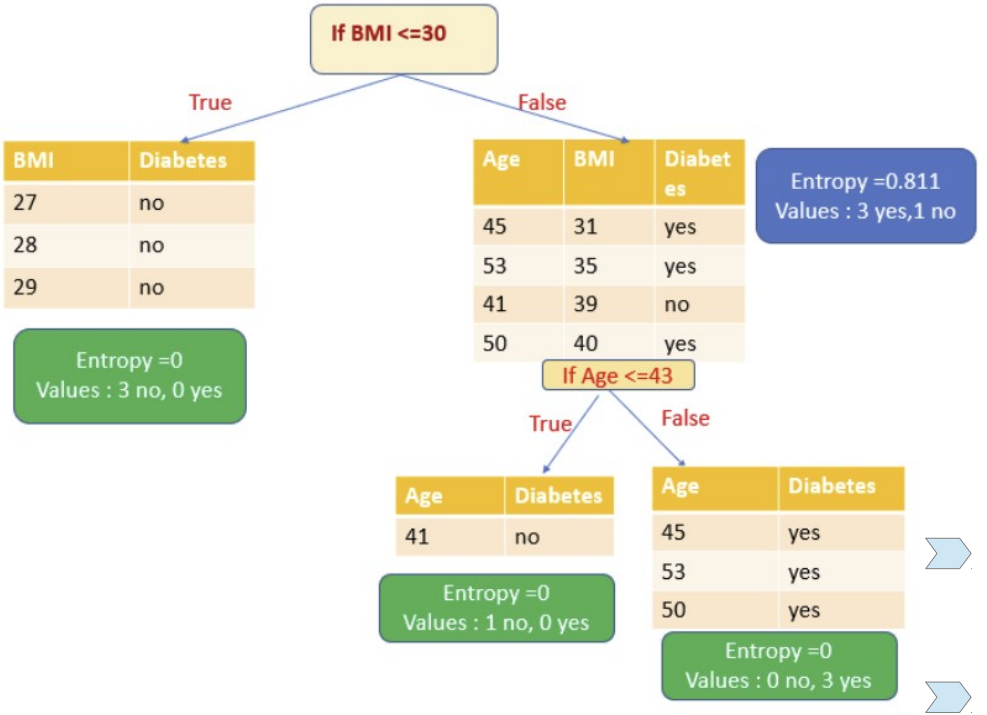
= 0.985-0.980

IG(Age) = 0.005

$$IG(D_p) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

- Choose the attribute with the largest $IG(A)$

How does splitting occur?



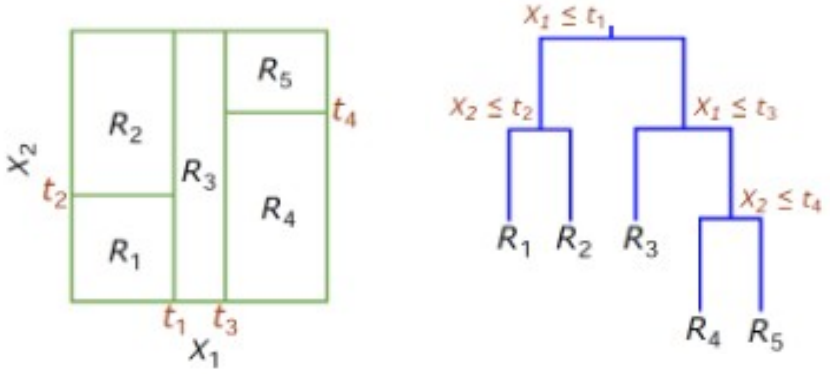
What is the optimal Tree Depth?

- We need to be careful to pick an appropriate **tree depth**.
- If the tree is too **deep**, we can **overfit**.
- If the tree is too **shallow**, we **underfit**
- **Max depth** is a hyper-parameter that should be tuned by the data.
- Alternative strategy is to create a very deep tree, and then to **prune** it.

You grow the tree entirely using your decision tree algorithm and then you and then prune it back in order to obtain a subtree.

You start from the bottom decision node and, based on measures such as Gini Impurity or Information Gain, you decide whether to keep this decision node or replace it with a leaf node.

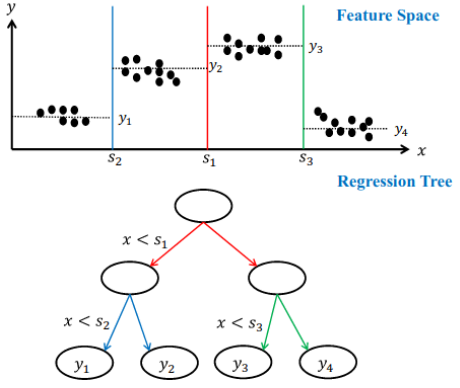
Regression Trees



- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.



In order to perform Recursive binary splitting, we select the predictor and cutpoint that leads to the greatest reduction i RSS.

$$R_1(j,s) = \{X | X_j \leq s\} \quad \text{and} \quad R_2(j,s) = \{X | X_j > s\}.$$

Figure-3: For any variable j and splitting point s

We seek the value of j and s that minimize the equation:

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2,$$

Figure-4: RSS of Recursive Splitting

Cost-complexity pruning

- When should tree growing be stopped?
- Will need to control complexity to prevent over-fitting, and in general find optimal tree size with best predictive performance.

- A better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*
- *Cost complexity pruning* — also known as *weakest link pruning* — is used to do this
- we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

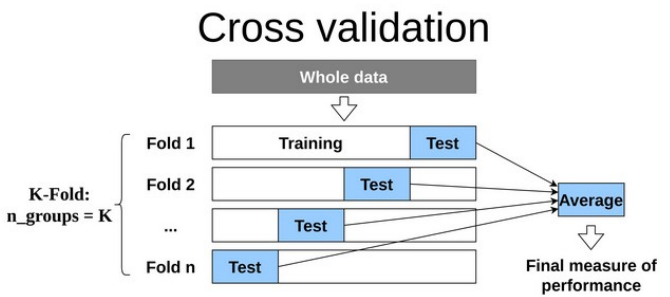
$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

K-fold cross-validation

Cross-validation means, that we create k type of subdivision between on the data. We train on each fold one model, and in the end, we calculate the average. Code for this [2]



http://mlwiki.org/index.php/Cost-Complexity_Pruning

Regression Tree algorithm

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

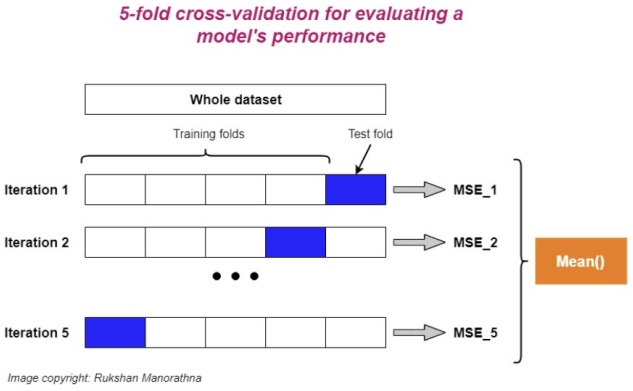


Image by Author

The general process of k-fold cross-validation for evaluating a model's performance is:

- The whole dataset is *randomly* split into *independent* **k-folds** *without replacement*.
- **k-1 folds** are used for the model training and one fold is used for performance evaluation.
- This procedure is repeated **k times** (iterations) so that we obtain **k number** of performance estimates (e.g. MSE) for each iteration.
- Then we get the mean of **k number** of performance estimates (e.g. MSE).

Bias Variance tradeoff

If we denote the variable we are trying to predict as Y and our covariates as X, we may assume that there is a relationship relating one to the other such as $Y = f(X) + \epsilon$ where the error term ϵ is normally distributed with a mean of zero like so $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$.

We may estimate a model $\hat{f}(X)$ of $f(X)$ using linear regressions or another modeling technique. In this case, the expected squared prediction error at a point x is:

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

This error may then be decomposed into bias and variance components:

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

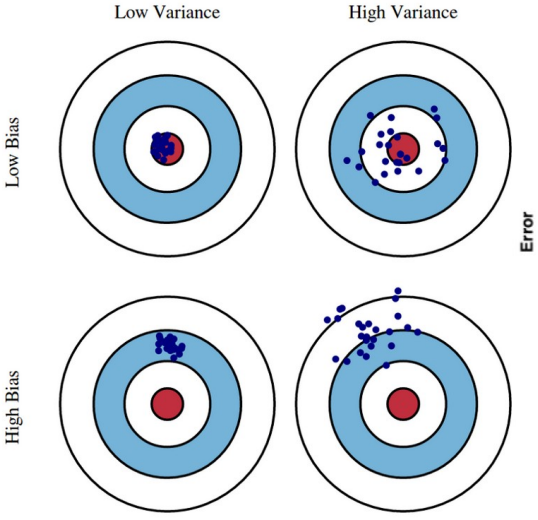
$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

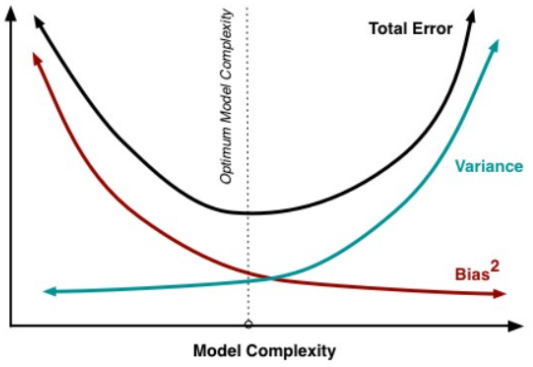
Bias

It refers to model fitting the training data **poorly** but able to produce similar result in data outside training data. It is related to **underfitting**.

Variance

We are building complex models that fits well on training data but they cannot generalise the pattern well which results to **overfitting**. It means they don't fit well on data outside training (i.e. validation / test datasets).





- Decision trees such as regression or classification trees suffer from high variance.
- This means that if we split the training data into two parts at random and fit a decision tree to both halves, the results can be quite different.

Bagging

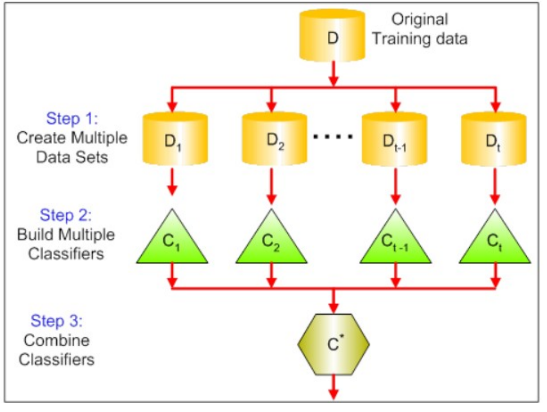
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.
- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

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

This is called *bagging*.

Bagging comes from the words ***B***ootstrap + ***AGG***regat***ING.*** We have 3 steps in this process.

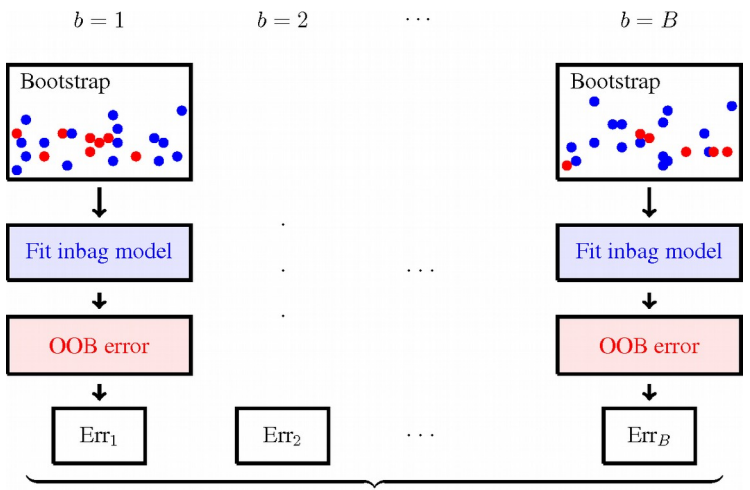
1. We take 't' samples by using row sampling with replacement (doesn't matter if 1 sample has row 2, there can be another sample with row 2 which are totally independent) from our original training data called D1,D2… Dt- Bootstrapping.
2. Once we have those samples, we build different classification models on each of them C1,C2…..Ct.
3. Now we combine these classifiers together by taking a majority vote for classification & mean/mode for regression problems- Aggregating



Bagging

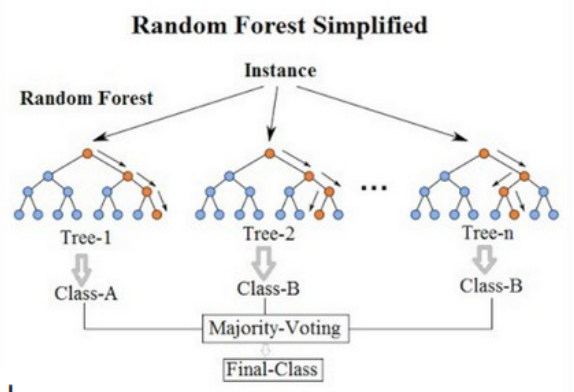
Test error estimation for bagging trees

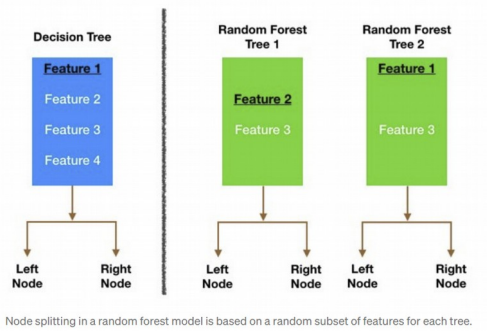
- Cross-validation or vaidation set
- Out-of-Bag (OOB) error estimation
  - One can show that on average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
  - We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation. In order to obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the ith observation.
- The OOB approach for estimating the test error is particularly convenient when performing bagging on large data sets for which cross-validation would be computationally onerous.



Random Forests

- The bagged trees based on the bootstrapped samples often look quite similar to each other. They are therefore often **Highly correlated**.
- averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quanti- ties. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.





Node splitting in a random forest model is based on a random subset of features for each tree.

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.
- As in bagging, 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.
- The split is allowed to use only one of those m predictors.

- For m, it is recommended that
  - For classification, the default value for m is $\sqrt{p}$ and the minimum node size is one.
  - For regression, the default value for m is $p/3$ and the minimum node size is five.

Random Forests

Advantages of using Random Forest technique:

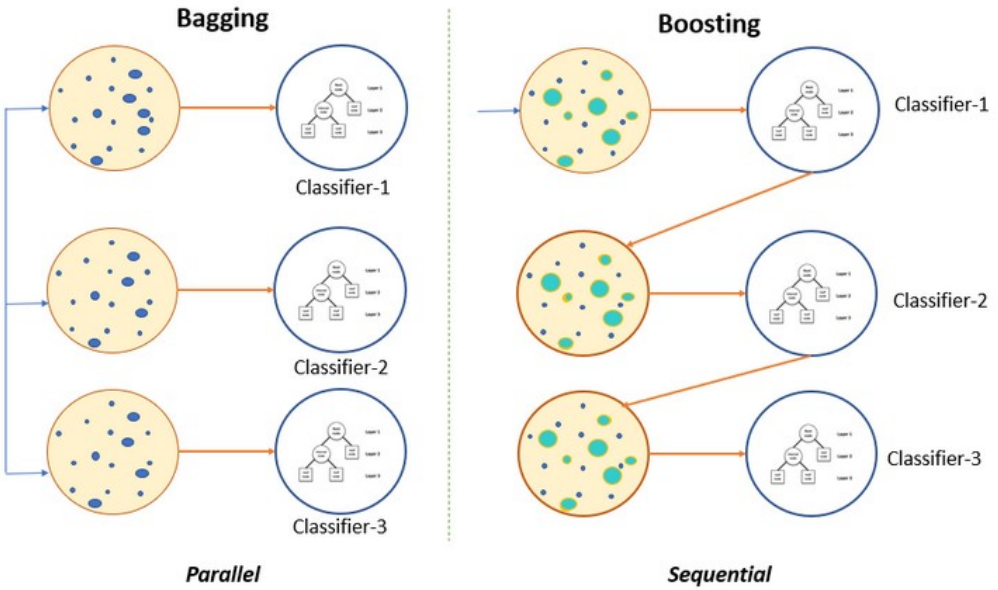
- Handles higher dimensionality data very well.
- Handles missing values and maintains accuracy for missing data.

Disadvantages of using Random Forest technique:

- Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the regression model.

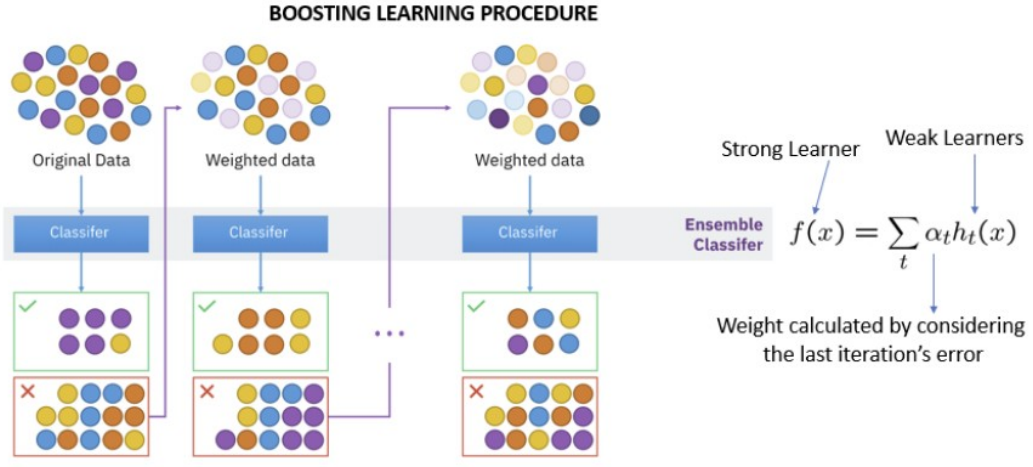
Boosting

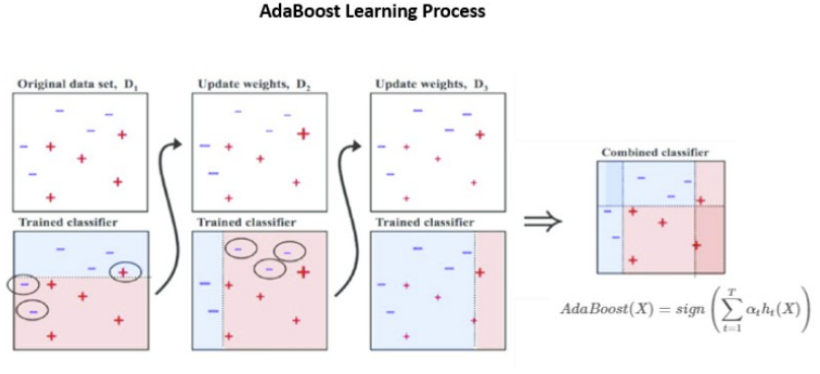
- Like bagging, boosting involves combining a large number of decision trees.
- Different from bagging, bossting trees are grown **sequentially**: each tree is grown using information from previously grown trees
- Boosting does not involve bootstrap sampling; instead each tree is fit on a weighted original dataset: **The samples that are misclassified by the previouse tree receive more weights**
- The individual trees do not contribute to the final prediction equally: **Give larger weights to more accurate classifiers in the sequence.**



Representative algorithms using boosting techniques are GradientBoost, AdaBoost, and XGBoost.

AdaBoost(Adaptive Boosting)



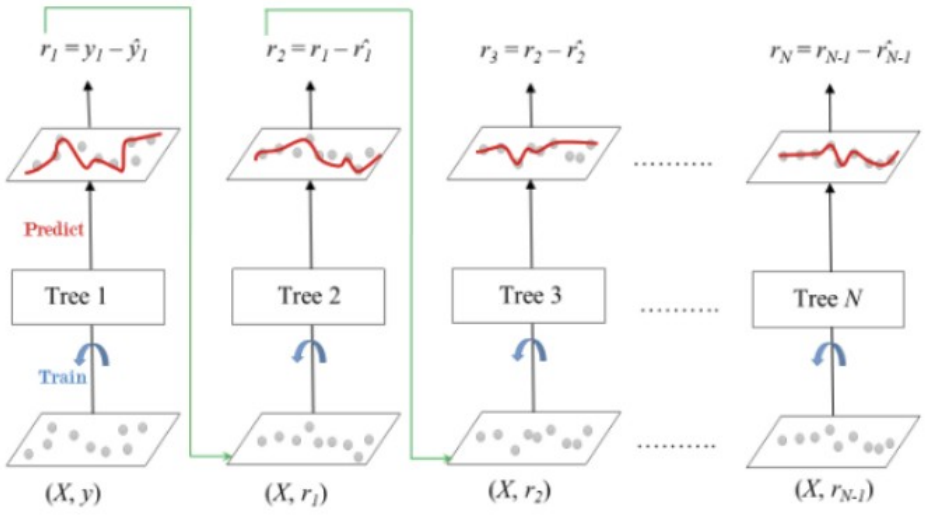


Algorithm 10.1 *AdaBoost.M1.*

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

Gradient Boosting

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.



Gradient Boosted Trees for Regression

Algorithm 8.2 *Boosting for Regression Trees*

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

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

XGBoost (Extreme Gradient Boosting)

- XGBoost or Extreme Gradient Boosting is an optimized implementation of the Gradient Boosting algorithm. Since its introduction in 2014 XGBoost has been the darling of machine learning hackathons and competitions because of its prediction performance and processing time.

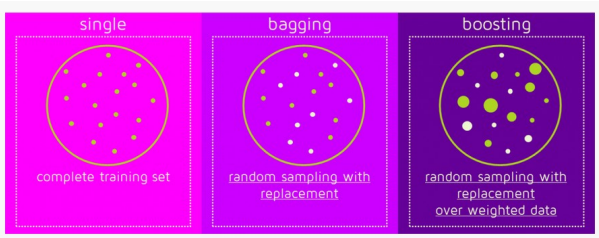
- XGBoost is more regularized form of Gradient Boosting. XGBoost uses advanced regularization (L1 & L2), which improves model generalization capabilities.
- XGBoost delivers high performance as compared to Gradient Boosting. Its training is very fast and can be parallelized / distributed across clusters.
- XGBoost computes second-order gradients, i.e. second partial derivatives of the loss function, which provides more information about the direction of gradients and how to get to the minimum of our loss function.
- XGBoost also handles missing values in the dataset. So, in data wrangling, you may or may not do a separate treatment for the missing values, because XGBoost is capable of handling missing values internally.

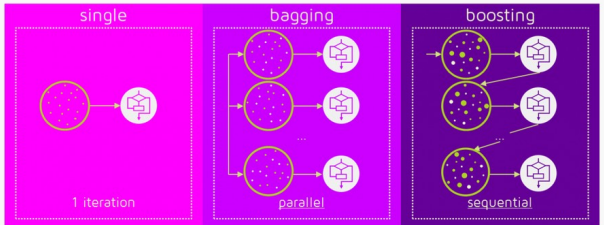
Tuning parameters for boosting

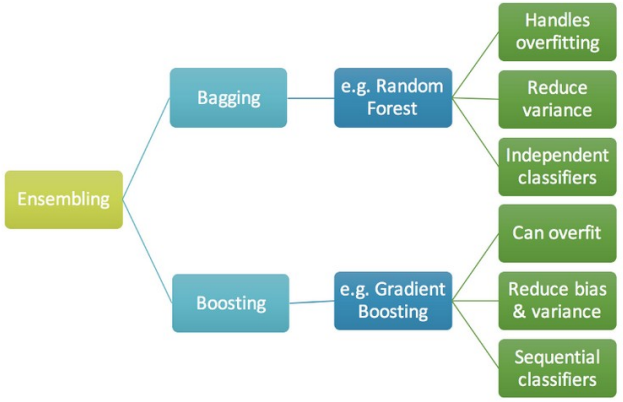
- The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The common practice is to restrict all individual trees to have the same size.
  - Let d be the number of splits in each tree. When $d = 1$, each tree is a stump , consisting of a single split. In this case, the boosted stump ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the **interaction depth**.
  - For regression, $d = 1$ often works well. For classiciation, experience indicates that $d \in [3, 7]$ works well in the context of boosting, with results being fairly insensitive to particular choices in this range. One can fine-tune the value for d by trying several different values and choosing the one that pro duces the lowest risk on a validation sample. However, this seldom provid es significant improvement over using $d \approx 5$.

- The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

Bagging vs Boosting







Advantages of Bagging

- The biggest advantage of bagging is that multiple weak learners can work better than a single strong learner.
- It provides stability and increases the machine learning algorithm's accuracy that is used in statistical classification and regression.
- It helps in reducing variance, i.e. it avoids overfitting.

Disadvantages of Bagging

- It may result in high bias if it is not modelled properly and thus may result in underfitting.
- Since we must use multiple models, it becomes computationally expensive and may not be suitable in various use cases.

Advantages of Boosting

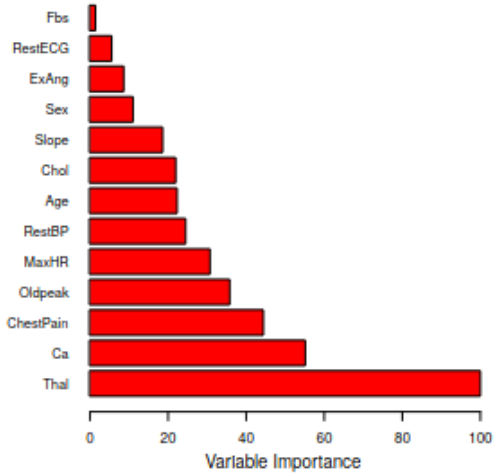
- It is one of the most successful techniques in solving the two-class classification problems.
- It is good at handling the missing data.

Disadvantages of Boosting

- Boosting is hard to implement in real-time due to the increased complexity of the algorithm.
- The high flexibility of these techniques results in multiple numbers parameters that directly affect the behaviour of the model.

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot for the `Heart` data

Boston.csv

- **LON** and LAT are the longitude and latitude of the center of the census tract.
- **MEDV** is the median value of owner-occupied homes, measured in thousands of dollars.
- **CRIM** is the per capita crime rate.
- **ZN** is related to how much of the land is zoned for large residential properties.
- **INDUS** is the proportion of the area used for industry.
- **CHAS** is 1 if a census tract is next to the Charles River else 0
- **NOX** is the concentration of nitrous oxides in the air, a measure of air pollution.
- **RM** is the average number of rooms per dwelling.
- **AGE** is the proportion of owner-occupied units built before 1940.
- **DIS** is a measure of how far the tract is from centres of employment in Boston.
- **RAD** is a measure of closeness to important highways.
- **TAX** is the property tax per $10,000 of value.
- **PTRATIO** is the pupil to teacher ratio by town.