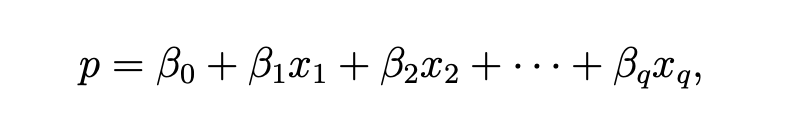
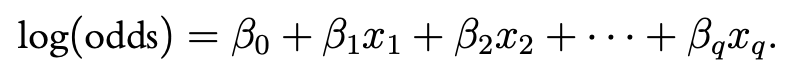
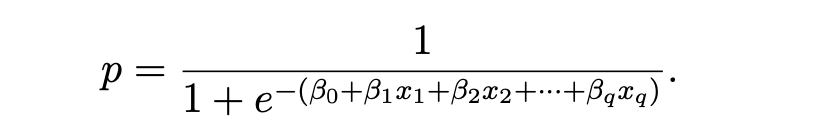
1. **Logistic regression**
   1. Logic
      1. From our basic linear regression, we get probability(p):



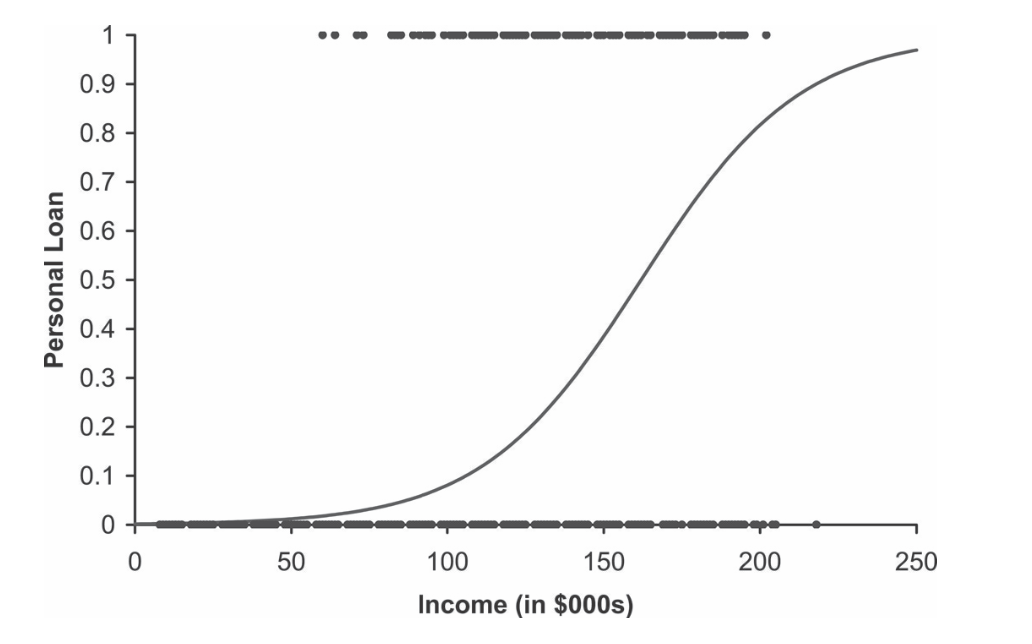
However, the estimated value of p will be out of range(0,1)-- not possible

* + 1. So, logistic regression make a transformation of this:





The first part is called logit in logistic regression(linear model in i.) and the second is called the logistic response function(as we will use type=response in R to get the probability of each record). Other things don’t change, that is we can always analyze the relationship between each independent variable(Xs) and target variable(Y).



* 1. Parameters need to be noticed
     1. Regularization parameter(penalty) : alpha=1/C

When we have too many variables(complicate model), we need to add parameter C in our logistic regression to exclude unimportant variables. The C smaller, the #variables will be smaller and the model will be simpler

* 1. Pros and Cons
     1. Pros:
        1. fast to train and to predict
        2. Work well will sparse data(have many meaningless values, like NA or 0)
        3. Easy to understand how a prediction is made
     2. Cons:
        1. Large dataset is needed
        2. Cannot deal with multicollinearity problem-- will cause bias on estimating coefficients
        3. Interpretation of coefficients is not intuitive
  2. Model evaluation
     1. **Confusion matrix**
        1. Accuracy: overall accuracy

(TP+TN) / All

* + - 1. Recall (sensitivity): True positive rate

TP / (all true positive)

* + - 1. Specificity: True negative rate

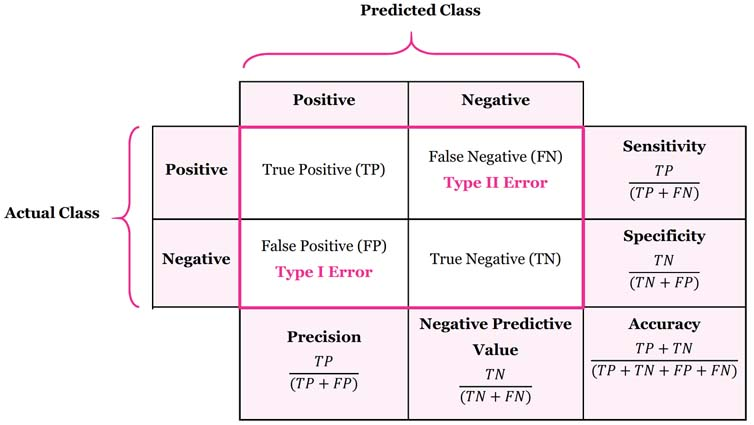
TN / (all true negative)

* + - 1. Precision: when predicted positive, how often is it correct?

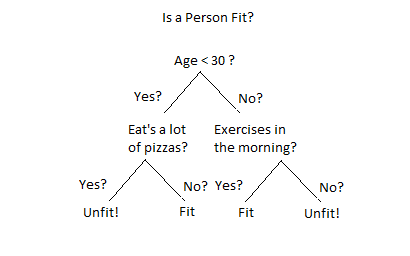
TP / (all predicted positive)

* + - 1. **F1-score**: weighted average of the true positive rate(recall) and precision

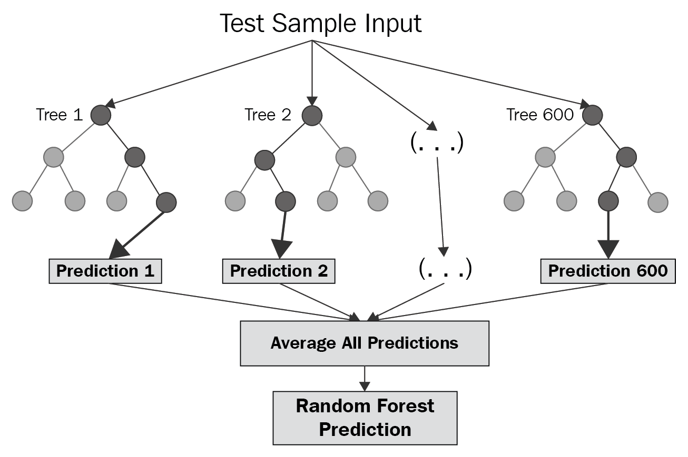
F1=2\*precision\*recall / (precision+recall)



1. **Random Forest**
   1. Logic
      1. Random Forest starts from a “decision tree”.



* + 1. A random forest is essentially a collection of decision trees, whereas each tree is slightly different from others. (bootstrap sampling)

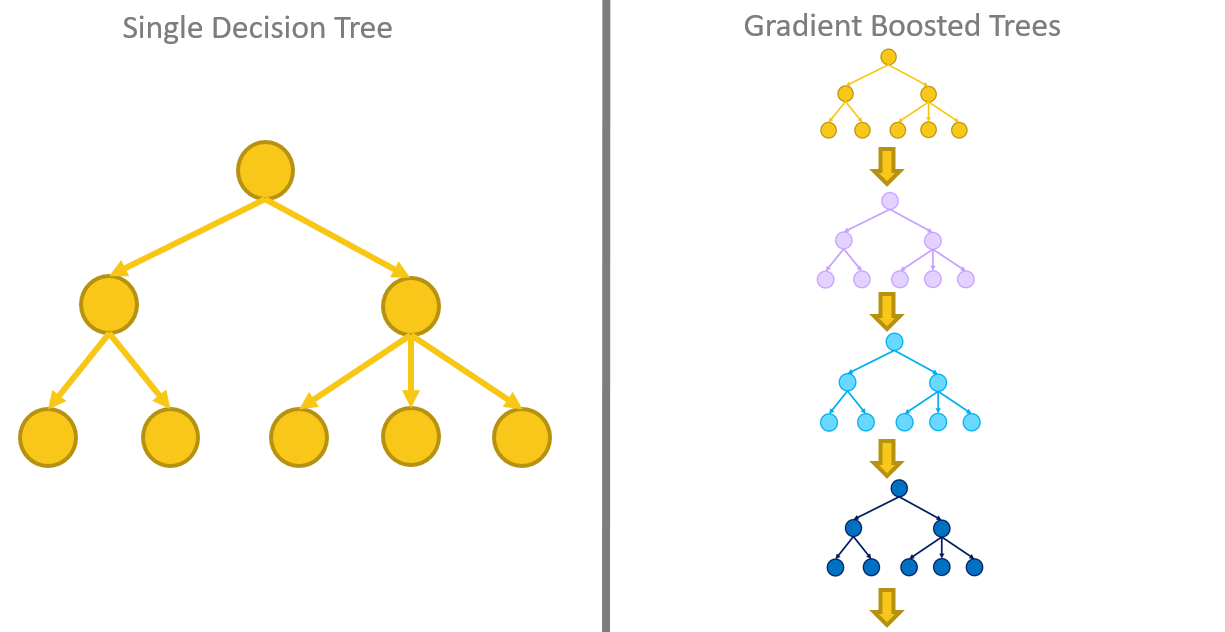


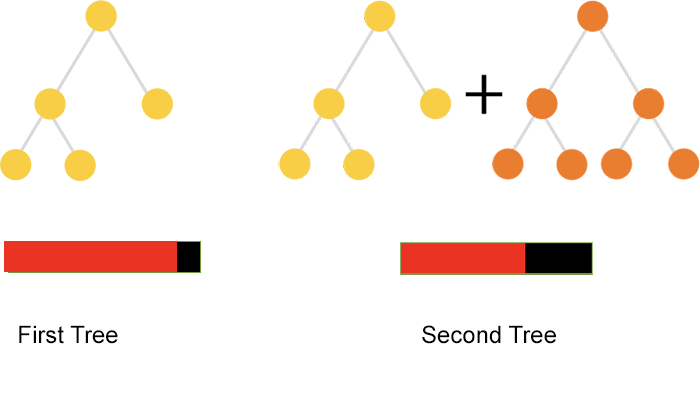
* 1. Parameters in the model
     1. Number of trees: Performance improves as the number of trees increases, then flattens out
     2. Max\_features: determines how random each tree is, smaller max\_features reduces overfitting,max\_features=sqrt(n\_features) for classification and max\_features=n\_features for regression.
     3. Tree\_depth: Probably close to optimal depth for a single tree
     4. Grid search for optimal parameters
  2. Pros and Cons
     1. Pros:
        1. Nearly always perform better than a single decision tree, very robust and powerful.
        2. Don’t need scaling of data.Can handle missing data.
        3. Handle big data with numerous variables running into thousands.
        4. Present estimates for variable importance.
     2. Cons:
        1. Not good for very high-dimensional sparse data
        2. Difficult to visualize compared with decision trees
  3. Random forest & Gradient Boosting Tree

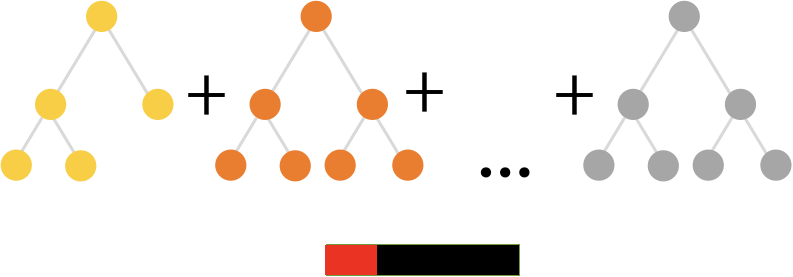
In contrast to the random forest approach, gradient boosting works by building trees in a serial manner, where each tree tries to correct the mistakes of the previous one.

1. **Gradient Boosting Tree**
   1. Logic:

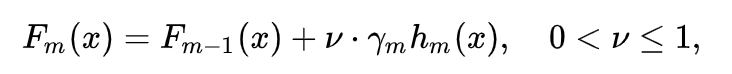
Produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees







* 1. Parameters:
     1. the number of gradient boosting iterations *M* (i.e. the number of trees in the model when the base learner is a decision tree)
     2. Learning rate “v”



Empirically it has been found that using small learning rates (such as v < 0.1) yields dramatic improvements in models' generalization ability over gradient boosting without shrinking (v = 1)

* + 1. Number of observations in leaves

The minimum number of observations in trees' terminal nodes

* 1. Pros and cons:
     1. Pros:
        1. Often provides predictive accuracy that cannot be beaten.
        2. Lots of flexibility - more parameters to be tuned
        3. No data pre-processing required - often works great with categorical and numerical values as is.
        4. Handles missing data - imputation not required.
     2. Cons:
        1. GBMs are more sensitive to overfitting if the data is noisy.
        2. Training generally takes longer because of the fact that trees are built sequentially.
        3. GBMs are harder to tune than RF

Gradient Descent (Statquest): <https://www.youtube.com/watch?v=sDv4f4s2SB8>

* can optimize all kinds of problems

Stochastic Gradient Descent (Statquest): <https://www.youtube.com/watch?v=vMh0zPT0tLI>

XGBoost (Statquest):

<https://www.youtube.com/watch?v=OtD8wVaFm6E>

<https://www.youtube.com/watch?v=8b1JEDvenQU>

<https://www.youtube.com/watch?v=ZVFeW798-2I>

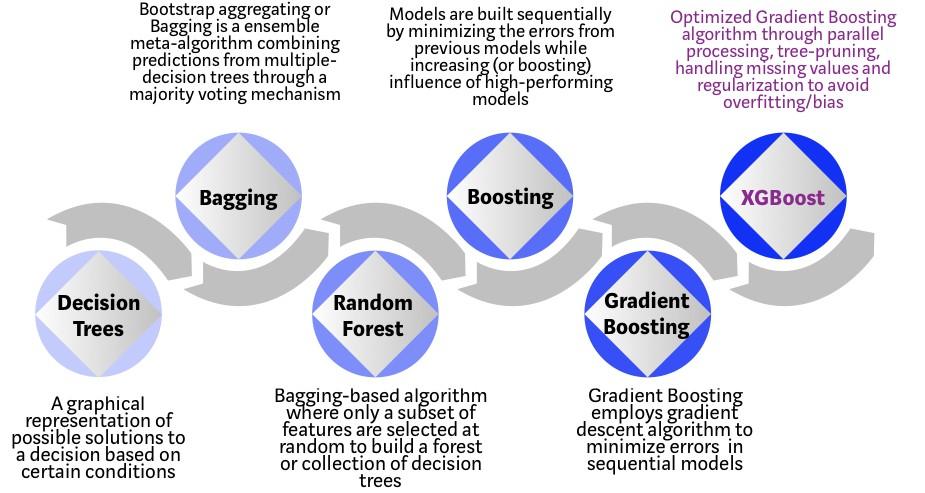
<https://www.youtube.com/watch?v=oRrKeUCEbq8>

**XGBoost(eXtreme Gradient Boosting)**

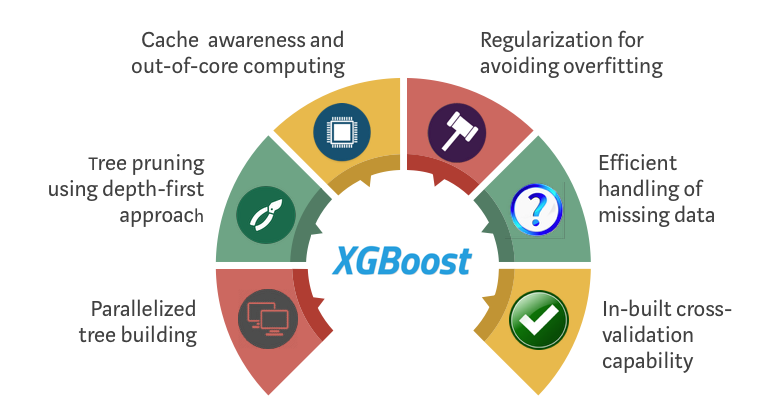
Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

XGBoost is a perfect combination of software and hardware optimization techniques to yield superior results using less computing resources in the shortest amount of time.



XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners ([CARTs](https://www.datasciencecentral.com/profiles/blogs/introduction-to-classification-regression-trees-cart) generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

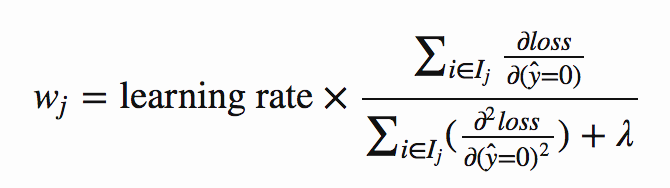


**Parameter**

**n\_estimators**: early stopping - prevent overfitting; not specified - return the lowest valid score one, specified - return the final model

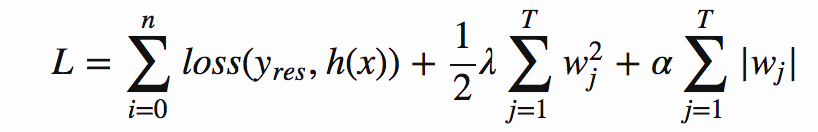
**max\_depth**: default is 3; never go beyond 5

**learning\_rate**: each weight ( in all trees) will be multiplied by this:



Decreasing it very often improves the performance of the model, but longer training time.

**reg\_alpha/ reg\_lambda**: control L1 L2 regularization terms - limit how extreme the weights at the leaves can become.



L2-lambda-weight to be small; L1-alpha-weight to go to zero-useful for feature selection.

Setting high lambda and low (or 0) alpha value is most effective when regularizing.