Decision Trees - Background

Ramesh Srinivasan

November 12, 2024

Decision Tree Models

- Formally, a decision tree model is one in which the final outcome of the model is based on a series of comparisons of the values of predictors against threshold
- In a graphical representation (flow chart),
  1. the internal nodes of the tree represent attribute testing.
  2. branching in the next level is determined by attribute value
  3. terminal leaf nodes represent class assignments.
- Decision Trees are like flowcharts that can be formulated as mathematical models for classification, with very desirable properties:
  1. Interpretable by humans.
  2. Allow for complex decision boundaries
  3. Decision boundaries are locally linear, hence simple to formulate
- Decision tress can be communicated as heuristics enhancing understanding.

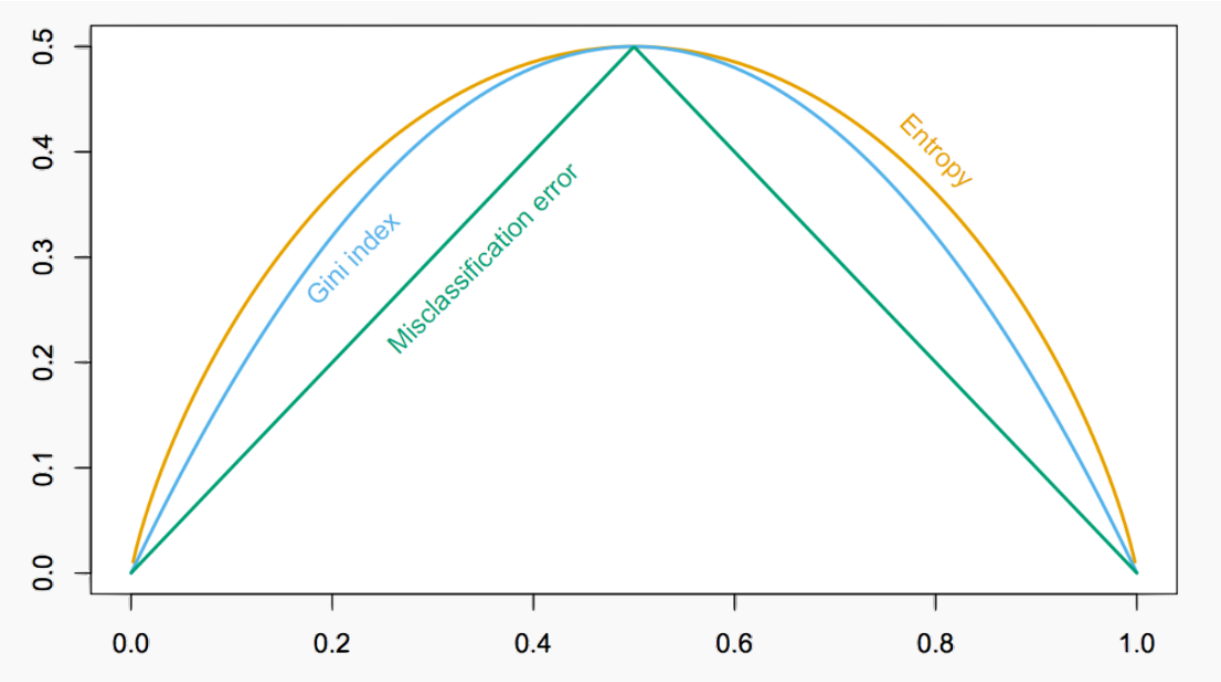
Flowcharts and Decision Trees

GRAB LEMON FIGURE HERE

- Flow charts whose graph is a tree (connected and no cycles) represents a model called a decision tree.
- Formally, a decision tree model is one in which the final outcome of the model is based on a series of comparisons of the values of predictors against threshold values.
- Learning a Decision Tree model:
  1. optimal partition of the feature-space with axis-aligned linear boundaries.
  2. each region gets a class label based on the largest class of training points in that region (Bayes' classifier)

Splitting Criteria

- The regions of the feature space should be as pure as possible without overfitting.
- If we can choose a fitness metric that is differentiable, optimization is easier
- You cannot have empty regions that do not contain training points
- Historically 3 approaches have been used
  1. Classification Error
  2. Gini Index
  3. Entropy



Gini Index

- Assume a K class prediction problem with J predictors and N data points.
- For the j^{th} predictor, splitting the region means finding a threshold t_j that is optimal
- for each new subregion R_i we can compute a Gini index which measures the purity of the region:

$$G(i|j, t_j) = 1 - \sum_k p(k|R_i)^2$$

- To decide how to split we find the predictor j and threshold t_j that minimizes the average Gini index weighted by the population of the regions.
- for a 2-class problem this would be

$$min_{j,t_j}(\frac{N_1}{N} G(1|j, t_j) + \frac{N_2}{N} G(2|j, t_j)$$

where N_1, N_2 are the number of points in each subregion and

$$G(1|j, t_j) = 1 - (p(1|R_1)^2 + p(2|R_1)^2$$

$$G(2|j, t_j) = 1 - (p(1|R_2)^2 + p(2|R_2)^2$$

Entropy

- For a random variable with a discrete distribution, entropy is

$$H(x) = -sum_{x \in X} p(x) log_2 p(x)$$

- For a discrete distribution, a flat histogram has higher entropy as all values are equally probable to be sampled from that distribution. A distribution with a strong peak has lower entropy.
- Suppose select a predictor j and split the N training points along a threshold t_j. We can measure the quality of the split by measuring the entropy of the class distribution in each new region.
- Then in a 2 class problem, the optimization problem becomes

$$min_{j,t_j} (\frac{N_1}{N} H(1|j,t_j) + \frac{N_2}{N} H(2|j,t_j)$$

Toy Problem - Play Tennis

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

We can calculate the overall entropy of playing tennis, Entropy for (9 Yes, 5 No)

$H(X) = -P(playtennis = yes)log_2(P(playtennis = yes) - P(playtennis = no)log_2(P(playtennis$

$$H(X) = -\frac{9}{14}log_2(\frac{9}{14}) - \frac{5}{14}log_2(\frac{5}{14}) = 0.94$$

That's pretty close to completely random. Entropy of a fair coin is 1. This means that a priori, if we use no features, it's hard to predict if we will play tennis or not.

Information Gain

$$Gain = H(N) - \sum_i \frac{N_i}{N} H(N_i)$$

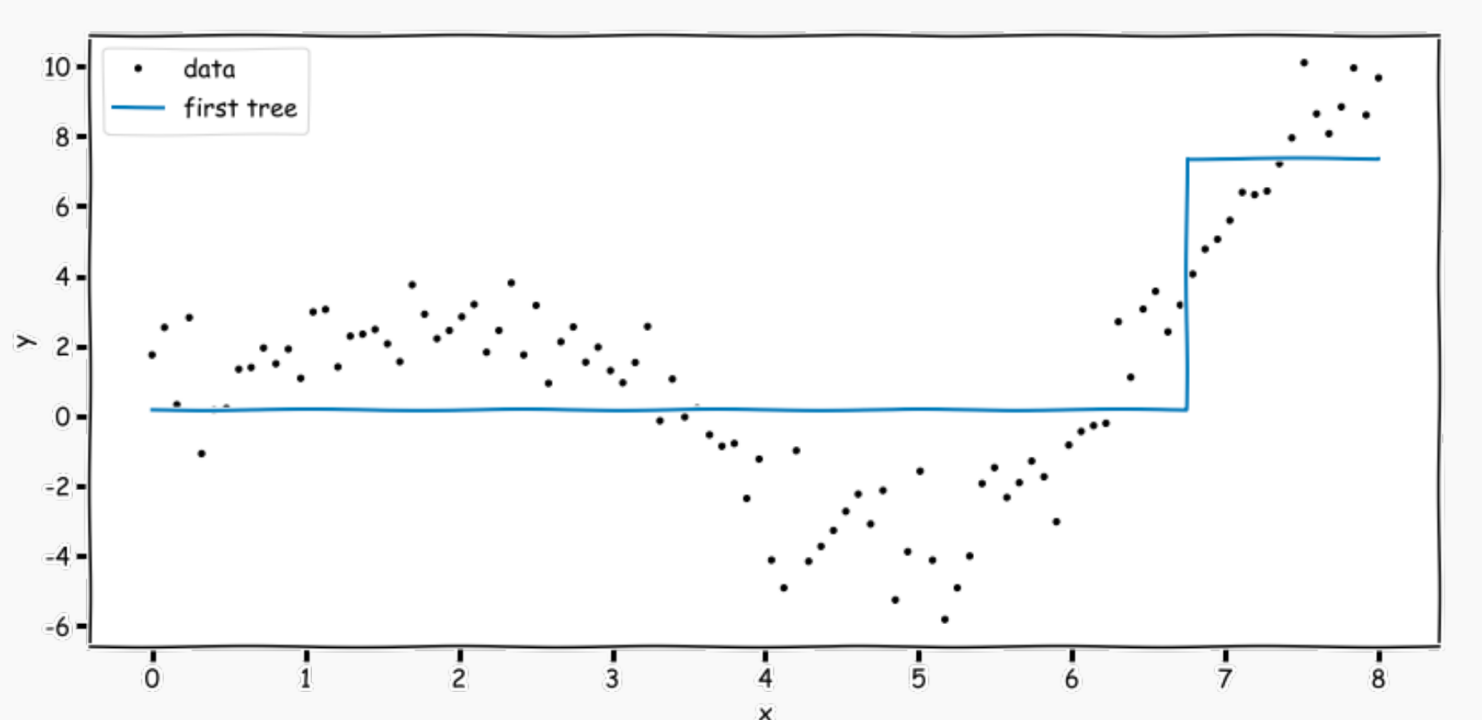
- if we divide the data into subsets using the values of a feature, we can compute the information gain.
- If we select 1 feature, e.g., Wind, there are two possible values , weak or strong. For weak subset there are 6+/2- for play tennis and for the strong subset there are 3+/3-
- The entropy for the weak subset it

$$H(x) = -\frac{6}{8} log_2(\frac{6}{8}) - \frac{2}{8} log_2(\frac{2}{8}) = 0.811$$

- The entropy for the strong subset is 1
- The information gain is

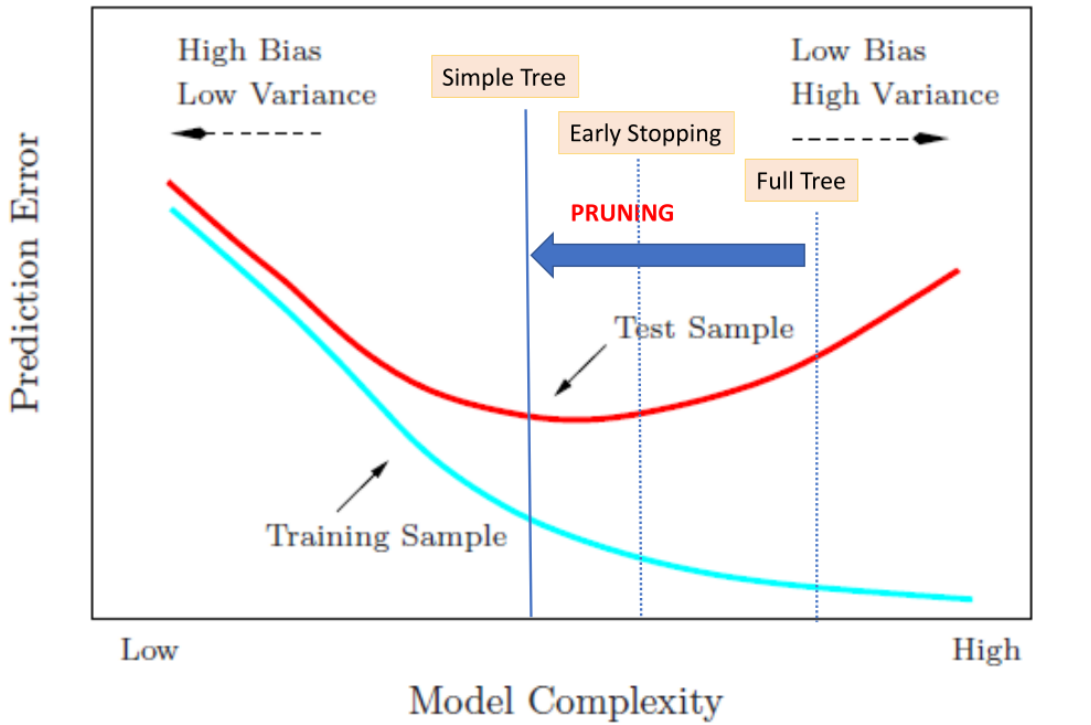
$$Gain = 0.94 - \frac{8}{14} 0.811 - \frac{6}{14} 1.00 = 0.048$$

Decision Tree Regression



The evaluative criteria is MSE

Overfitting



Approaches

- The greedy optimization (pruning) approach to determine parameters like depth doesn't scale particularly well.
- We will discuss some approaches here to improve the performance of these models.
- Although we will discuss the ideas here in the context of Decision Trees, the ideas are actually pretty useful when working with neural networks.
- One useful approach to circumvent this problem is Bootstrap Aggregation, or Bagging. This leads to a method called Random Forests which is widely used.
- The other approach is gradient boosting.

Bagging

- One way to adjust for the high variance of the output of an experiment is to perform the experiment multiple times and then average the results.
- The same idea can be applied to high variance models:
  1. **Bootstrap** we generate multiple samples of training data, via bootstrapping. We train a full decision tree on each sample of data.
  2. **Aggregate** for a given input, we output the averaged outputs of all the models for that input. For classification, we return the class that is outputted by the plurality of the models. For regression we return the average of the outputs for each tree.
- This method is called Bagging (Breiman, 1996), short for, of course, Bootstrap Aggregating

Advantages and Limitations

- The advantage of bagging is that
  1. High expressiveness - by using full trees each model is able to approximate complex functions and decision boundaries.
  2. Low variance - averaging the prediction of all the models reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees.
- The most important limitation of bagging is that we lose the simple interpretability of decision trees and have to use somewhat more complex interpretation methods
- In practice, the ensembles of trees in Bagging tend to be highly correlated. The greedy learning algorithm ensures that most of the models in the ensemble will choose to split on strongest predictors in early iterations.
- Then, each tree in the ensemble can be identically distributed, with the expected output of the averaged model the same as the expected output of any one of the trees.

Random Forests

- Random Forest is a modified form of bagging that creates ensembles of bootstrap sampled decision trees.
- To de-correlate the trees, we:
  1. train each tree on a separate bootstrap sample of the full training set (same as in bagging)
  2. for each tree, at each split, we randomly select a set of J' predictors from the full set of predictors.
  3. From the subset J' of predictors, we select the optimal predictor and the optimal corresponding threshold for the split.
- Then just like bagging the plurality vote for a classifier decides the input sample classification.

Tuning Random Forests

- Random Forest Models have multiple hyperparameters that could be optimized
  1. the number of predictors to randomly draw
  2. the total number of trees in the ensemble
  3. the minimum leaf node size to stop splitting.
- This works well but is computationally expensive
- There are default (recommended) values for these parameters, for example for the number of predictors to draw choose, $J' = \sqrt{J}$
- Random forest sucks if many predictors are useless, as the random sampling aspect of this, leaves you stuck with many useless trees.
- Increasing the number of trees does not increase the risk of overfitting.

Bagging/Random Forests versus Boosting

- A weak learner is a constrained model (e.g., limit the max depth of each decision tree).
- Bagging and Random Forest: -
  1. complex and deep trees overfit
  2. perform variance reduction by building many weak learners(simple, shallow trees) and aggregating them.
- Boosting:
  1. simple and shallow trees underfit
  2. let's perform bias reduction of simple trees by strategically making them more expressive (complex).
  3. a large number of weak learners in sequence.
  4. Each one in the sequence focuses on learning from the mistakes of the one before it
  5. By more heavily weighting in the mistakes in the next tree, our next tree will learn from the mistakes.
  6. A combining all the weak learners into a single strong learner a **boosted tree**.

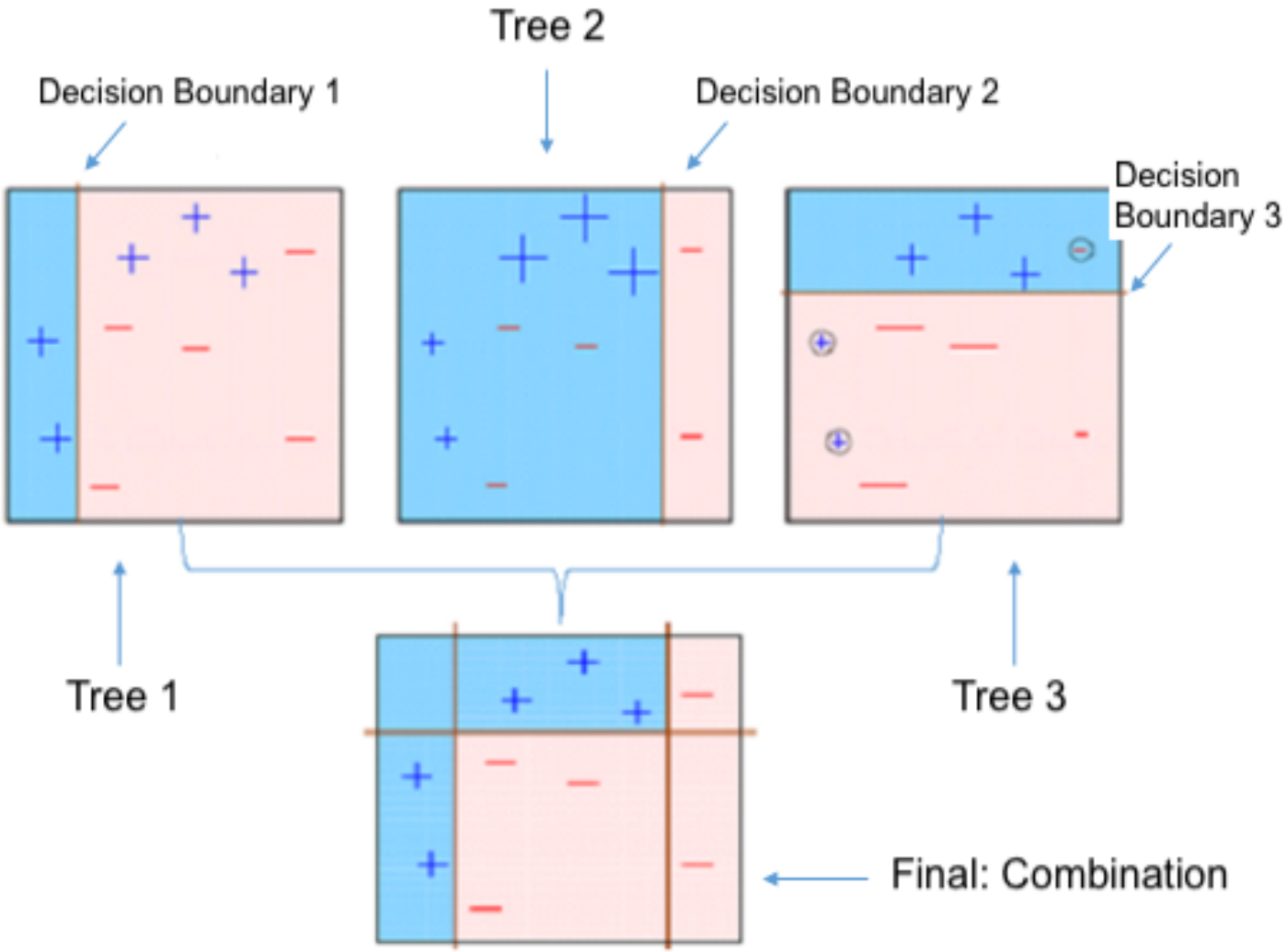
Boosting

- The key intuition behind boosting is that one can take an ensemble of simple models $T_h, h \in H$ and additively combine them into a single, more complex model T
- Each model is a poor fit to the data (weak learner) but a linear combination of the ensemble

$$T = \sum_h \lambda_h T_h$$

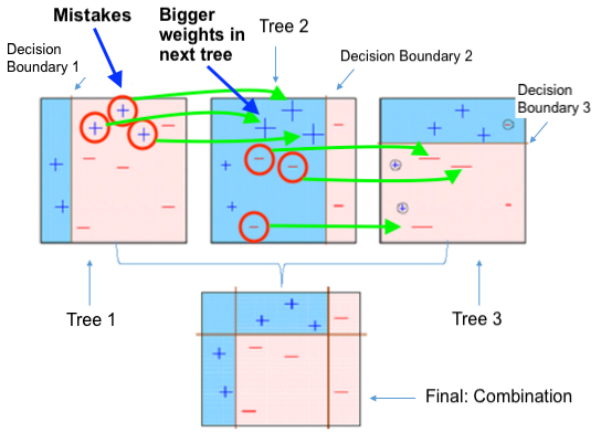
is a complex model sufficiently expressive to perform well.

Boosting


Tree 2
Decision Boundary 1
Decision Boundary 2
Decision
Boundary 3
Tree 1
Tree 3
Final: Combination

Boosting - Reweighting the Errors

- The size of the plus or minus signs indicates the weights of a data points for every Tree.
- For each consecutive tree and iteration we do the following:
  1. The wrongly classified data points are identified and more heavily weighted in the next tree.
  2. This change in weights will influence and change the next simple decision tree
  3. The correct predictions are identified and less heavily weighted in the next tree.
  4. We iterate this process for a certain number of times, stop and construct a linear combination of the simple trees, and is more expressive!



Gradient Boosting for Regression

- Fit a simple model $T^{(0)}$ to the training data $\boldsymbol{x}$, and $\boldsymbol{y}$. set $T \leftarrow T^{(0)}$
- Compute the residuals $\boldsymbol{r}$
- Fit a simple model $T^{(1)}$ between $\boldsymbol{x}$ and $\boldsymbol{r}$
- set $T \leftarrow T + \lambda T^{(1)}$
- compute residuals repeat until a stopping criterion is reached.

Gradient Boosting is Gradient Descent

- Intuitively, each simple model $T^{(i)}$ that we add to our ensemble model T, models the errors of T.
- In the regression case, at step i the residual is reduced as

$$r_n - \lambda T^{(i)}(x_n)$$

- You might notice that we are doing here is gradient descent. That is we are taking a step in the direction of reducing the error by training a simple model, with a learning rate of λ.
- In the case of regression, the weighting is embodied in the magnitude of the residuals.

Ada Boost

- For classifiers, we cant simply use classification error as our optimization function for gradient descent (not differentiable).
- Instead we replace the Error function with a differentiable function that is a good indicator of classification error

$$L = \frac{1}{N}\sum_{n=1}^{N} exp(-y_n\hat{y}_n)y_n in\{-1, 1\}$$

- This exponential loss function is an upper bound for error.
- The gradient is simply

$$\nabla_{\hat{y}} L = [-y_1 exp(-y_1\hat{y}_1), ..., -y_n exp(-y_1\hat{y}_n)$$

- We can see that the data gets reweighted as $w_n y_n$, where $w_n = exp(-y_n\hat{y}_n)$
- On each iteration the correct responses are weighted as e^{-1} while the incorrect responses are weighted as e

AdaBoost

- In the initial fit, weight all the data equally.
- Fit a simple classifier.
- update the weights depending on the classifier performance as

$$w_n \leftarrow \frac{w_n exp(-\lambda_{(i)} y_n T_{(i)}(x_n))}{Z}$$

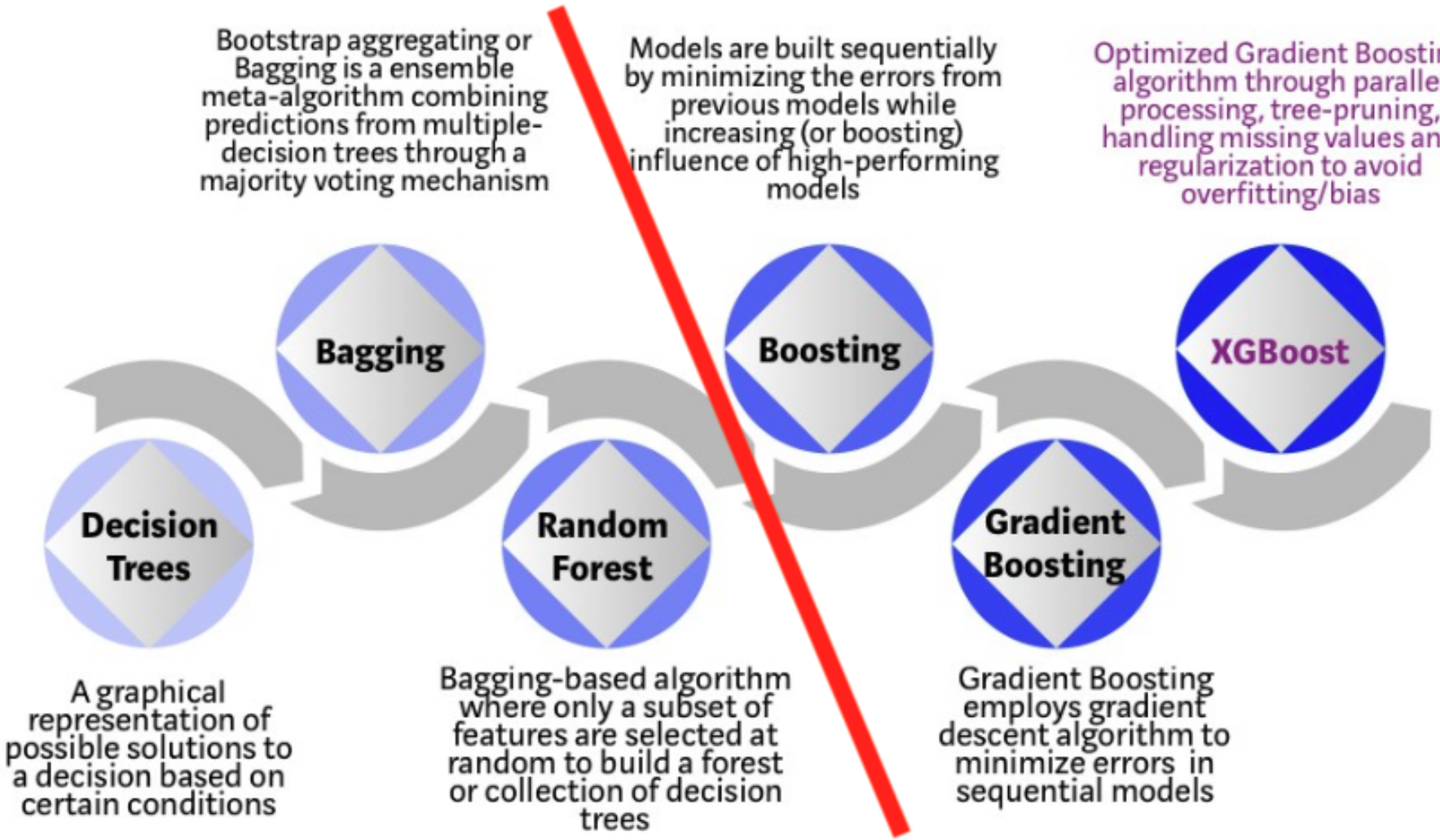
  where Z is a normalizing constant for the updated weights.
- update $T \leftarrow T + \lambda_{(i)} T_{(i)}$
- The optimal learning rate for AdaBoost on each step can be obtained analytically as

$$argmin_\lambda \frac{1}{N} \sum_{n=1}^{N} exp(-y_n(T + \lambda_{(i)} T^{(i)}(x_n))$$

$$\lambda^{(i)} = \frac{1}{2} ln \frac{1 - \epsilon^{(i)}}{\epsilon^{(i)}}$$

  where $\epsilon^{(i)}$ is the sum of the weights of the incorrect data.

XGBoost



Evolution of XGBoost Algorithm from Decision Trees

Summary

- The bias of a model quantifies how precise a model is across training sets.
- The variance quantifies how sensitive the model is to small changes in the training set.
- A robust model is not overly sensitive to small changes.
- The dilemma involves minimizing both bias and variance; we want a precise and robust model. Simpler models tend to be less accurate but more robust. Complex models tend to be more accurate but less robust.
- **How to reduce bias:**
  1. Use more complex models, more features, less regularization
  2. Boosting attempts to improve the predictive flexibility of simple models. Boosting uses simple base models and tries to "boost" their aggregate complexity.
- **How to reduce variance:**
  1. Early Stopping Rules provide us with guidance as to how many iterations can be run
  2. Pruning removes the nodes which add little predictive power for the problem
  3. Regularization introduces a cost term for bringing in more features with the objective function.
  4. Bagging tries to "smooth out" their predictions over random samples.
  5. Ensembling combining predictions from multiple separate models.