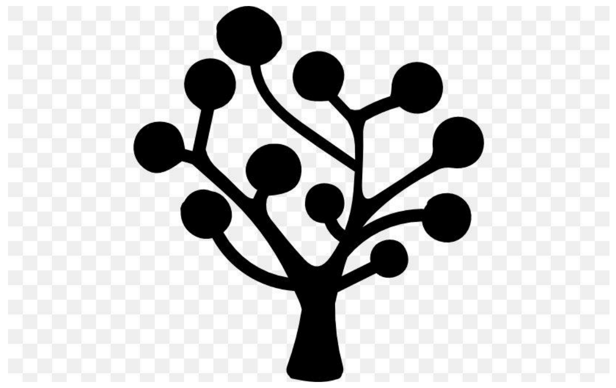


Decision Tree Cheat Sheet

V2020.12.14

(Dr Yan Xu)

Motivation

1. Similar inputs have similar outputs
2. Similarity determined by a tree structure
   - could be considered as a faster and smarter K-NN

Goal

1. Minimize impurity (or loss) of leaves
2. Minimize # of leaves

Note: *Finding an optimal tree is NP-Hard*

Impurity (or Loss) measurement

1. Classification
   - Gini impurity:

$$G(S) = \sum_{k=1}^{c} p_k(1 - p_k)$$

   where, p_k is fraction of inputs with label k

   - Entropy (or, KL -Divergence to Uniform):

$$\sum_{k} p_k log(p_k) + p_k log(c)$$

   where, *c* is number of classes

2. Regression
   - Squared Loss:

$$L(S) = \frac{1}{|S|} \sum_{(x,y)\in S} (y - \bar{y}_S)^2$$

Solution: Greedy

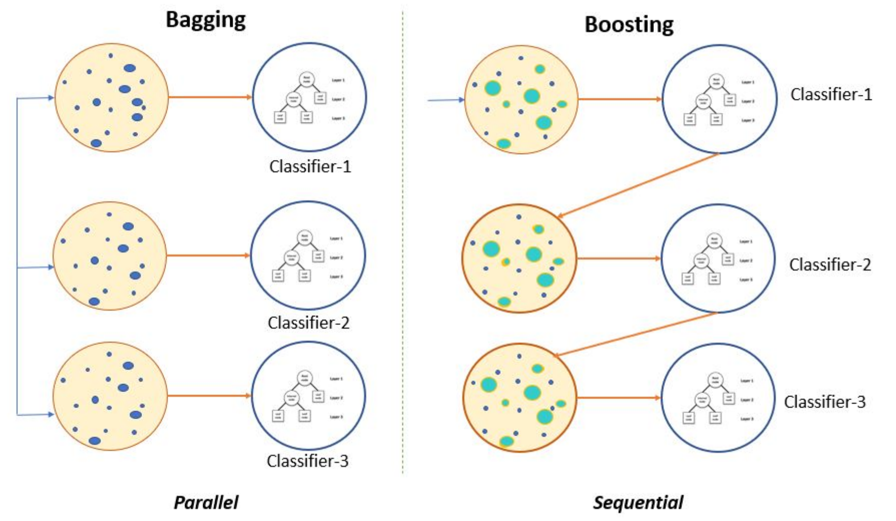
1. A top-down iterative split method
2. For each iteration, the split with the minimum impurity/loss is chosen
3. May stop early for some reasons (e.g. max depth)

Tree Ensembles

➔ One tree is not yet powerful, but ensembling trees is a great idea.

➔ Popular Solutions (Regression & Classification):
   - Random Forest
   - Gradient Boosting with Regularization (or XGBoost, LightGBM, etc)

➔ Bagging vs Boosting



(source: https://www.pluralsight.com/guides/ensemble-methods:-bagging-versus-boosting)

Random Forest

➔ Bagging is a general method to deal with unstable predictions, so does Random Forest

➔ Original Algorithm [paper]:

1. Sample m data sets $D_1, \ldots, D_m$ from D with replacement.
2. For each D_j train a full decision tree $h_j()$ (max-depth=∞) with one small modification: before each split randomly subsample $k \leq d$ features (without replacement) and only consider these for your split. (This further increases the variance of the trees.)
3. The final classifier is $h(\mathbf{x}) = \frac{1}{m}\sum_{j=1}^{m} h_j(\mathbf{x})$.

➔ Bootstrap Sample (with replacement) is used. Why? I am not sure.

➔ Subsampling features is optional.

➔ The number of trees is critical, but not sensitive, so typically easy to choose.

➔ Libraries: scikit-learn

➔ Variance Prediction: forestci
   - Be careful: it is sampling variance

➔ ML Code Example:
   - RandomForest

➔ Model Tuning: Easy

Gradient Boosting

➔ Residual Boosting

➔ For Square Loss, Gradient = Residual

➔ For other cost functions, Gradients are viewed as Residuals' generalization.

➔ Algorithm with Square Loss:

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

```
start with an initial model, say, $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
    calculate negative gradients $-g(x_i)$
    fit a regression tree $h$ to negative gradients $-g(x_i)$
    $F := F + \rho h$, where $\rho = 1$
```

➔ Algorithm with General Cost Function

```
start with an initial model, say $F(x) = \frac{\sum_{i=1}^{n} y_i}{n}$
iterate until converge:
    calculate negative gradients $-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$
    fit a regression tree $h$ to negative gradients $-g(x_i)$
    $F := F + \rho h$
```

GBM with Regularization (XGBoost)

➔ Regularized Cost Function:

$$\mathcal{L}(\phi) = \sum_{i} l(\hat{y}_i, y_i) + \sum_{k} \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

➔ Second Order Approximation
   - faster convergence

➔ Shrinkage (or Step Size)

➔ Split Candidates: Quantile sketch

➔ Missing Value
   - default direction with higher gain

➔ Parallel & Distributed Computing

➔ Other Libraries:
   - LightGBM, CatBoost

➔ ML Code Example:
   - XGBoost

➔ Model Tuning: Not Trivial