

Pruning the tree

Pruning is the process of cutting down excess branches in order to give accurate results.

Objective of any Machine Learning model is - to train the model on the training data & predict on the test data with as much accuracy as possible.

But this is not always practical.

In some cases, model give good results with training data but doesn't give good results with testing data. This situation is called "overfitting". In cases, when model doesn't perform well on training it is called "underfitting".

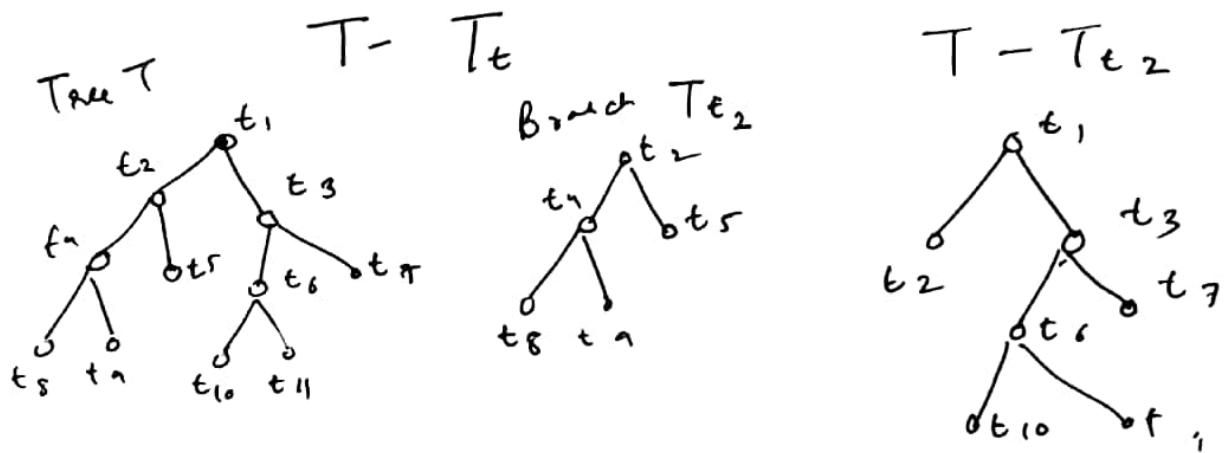
Pruning helps to avoid the problem of overfitting.

In some cases data might be missing. In that case one can simply repeat the max. occurring value or one can do a more sophisticated method - where one can repeat the already available data (numbers) based on the probability of the

Pruned

Pruning a branch T_t from a tree consisting of deleting from T all descendants of t , i.e. cutting off all T_t except its root node.

The tree pruned this way is described by



$$T' = T - T_{t_2}$$

T' is called pruned subtree of T

Optimal Subtree

Even for a moderately size of T_{max} ; there is enormously large no. of subtrees + an even larger no. ways to prune the initial tree to get any.

Tedious job to go through all subtrees to find out which one is best in some sense

One let a data point pass down the tree and see which leaf node it lands on.

Class of the leaf node is assigned to that new data point.

A class assignment rule assign a class $j = 1, \dots, K$ to every leaf node $t \in \tilde{T}$.

Class assigned to node t .

\tilde{T} is denoted by $k(t)$.

If $k(t) = 2$; all points in node t would be assigned to class 2.

If one use 0-1 loss; one pick the majority class or class with maximum probability.

$$k(t) = \arg \max_j p(j|t)$$

$$\begin{aligned} \arg \max(g(x)) &= x \\ x, g, b \\ \theta &= (x, g, b) \\ \hat{\theta} &= (0.3, 0.1, 0.6) \\ \arg \max(\hat{\theta}) &= b \end{aligned}$$

If one has to estimate the classification error rate for the tree for particular tree.

One introduce resubstitution estimate $\pi(t)$ for the probability of misclassification, given that a case falls into node t .

$$\pi(t) = 1 - \max_j p(j|t) = 1 - p(k(t)|t)$$

$$R(t) = \pi(t) p(t)$$

$\pi(t) \rightarrow 1 - \text{probability of majority class in node } t \text{ based on training data}$

$$R(T) = \sum_{t \in T} R(t)$$

L_i

If one splits a node into child nodes, misclassification rate is ensured to improve.

One requirement

One requires smaller way to prune.

- Subtree is optimal in certain sense.
- Search of the optimal subtree should be computationally easy to control

Let expected misclassification rate of a tree T be $R^*(T)$

$$R(T) = \sum_{t \in T} r(t) p(t)$$

$r(t)$ → probability of making wrong classification for point in node t .

For a point in a given leaf node t , estimated probability of misclassification is $1 - \text{probability of the majority class in node } t$ based on training data.

To get probability of misclassification for the whole tree, a weighted sum of within leaf node error rates is computed to the total probability.

No. of Terminal Node	$R(T)$
71	0.00
63	0.00
58	0.03
40	0.10
34	0.12
19	0.29
10	0.29
9	0.32
7	0.41
6	0.46

Tree is gradually pruned.

Resubstitution error rate $R(T)$ becomes monotonically larger when tree shrinks

$R(T)$ is not a good measure for selecting a subtree because it always favors bigger trees.

One need to add a complexity penalty to this resubstitution error rate.

Penalty term favors smaller tree & hence balances with $R(T)$

for any subtree $T \subset T_{\max}$, one defines its complexity as $|T|$, the no. of terminal or leaf nodes in T .

let $\alpha \geq 0$ be a real no. called complexity parameter + define the cost-complexity measure $R_\alpha(T)$ as

$$R_\alpha(T) = R(T) + \alpha(|T|)$$

$$\alpha(|T|) = \alpha \cdot |f(T)|$$

$f(T) \rightarrow$ fn that returns set of leaves of tree T



The more leaf nodes that the tree contain the higher complexity of the tree \therefore one has more flexibility in partitioning the space into smaller + \therefore more possibility for fitting training data.

One can fine tune the importance to be attached to the size of tree. This is done by complexity parameter α .

$$\alpha = \frac{R(+)-R(T_e)}{|f(T_e)|^{-1}}$$

If $\alpha=0$; biggest tree is chosen.

$\alpha \rightarrow \infty$; tree size decreases.

given α ; one needs to find subtree $T(\alpha)$ that minimizes $R_\alpha(T)$

$$R_\alpha(T(\alpha)) = \min_{T \leq T_{\max}} R_\alpha(T)$$

This method is called Minimal Cost - Complexity Pruning

Scikit Learn has this.

One can simply look at the example at Scikit Learn.

Best pruned subtree

- Use test sample set
- In case of less data use cross validation

Once the trees + subtrees are obtained; to find the best one out of them is computationally light.

Improving performance of decision tree

The performance of decision tree can be enhanced by aggregating many decision trees, using methods like

- bagging
- random forest
- boosting

Bagging

Decision trees suffer from high variance. If the training data is randomly divided into two part and a decision tree is created for both the part then the two decision tree will be quite different. Ideally, we should get similar results when algorithm is applied repeatedly to distinct dataset.

Bagging or Bootstrap aggregation is used to reduce the variance of prediction by combining the result of multi classifier modeled on different sub-samples of the same dataset.

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

↑
total no. of
Baggy

↓
predictor
individual
tree

In case of the classification problem.

Each tree in the forest predicts the category to which the new data point belongs.

Finally, the new data is assigned to the category that wins the majority.

Random Forest

Random Forest is a versatile ML technique that can perform both regression & classification

Bagged Trees

- reduce variance by averaging the ensemble result
- Allows trees to grow without pruning, reducing the tree depth size, resulting in high variance but lower bias, which can help improve predictive power
- Bagging trees allow the trees to have the final model which has used the entire feature space when considering node split

Limitation

Uses entire feature space when creating splits in trees. Suppose some variables within the feature space are indicating certain predictions, there is a risk of having forest of correlated trees, which increases bias & reduce variance.

Random Forest

Random Forest is a versatile ML technique that can perform both regression + classification. It gives better performance than decision trees as it does everything for you reducing no. of dimensions, treating missing values, outlier values & exploring data. It's better than bagged trees as it de-correlates trees. As in bagging, R.F. creates no. of decision trees on training dataset.

For creating these trees, each split consider a random sample of m predictors as split candidates from the full set of predictors.

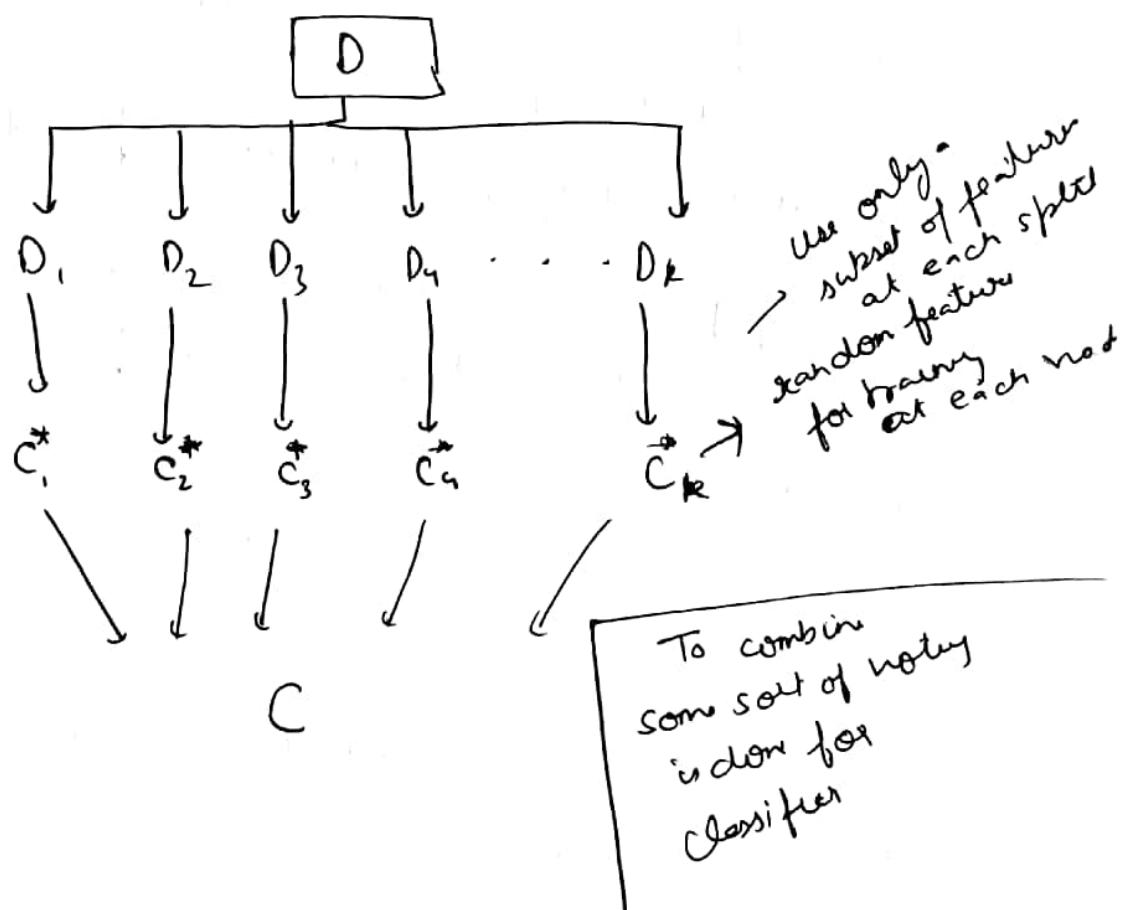
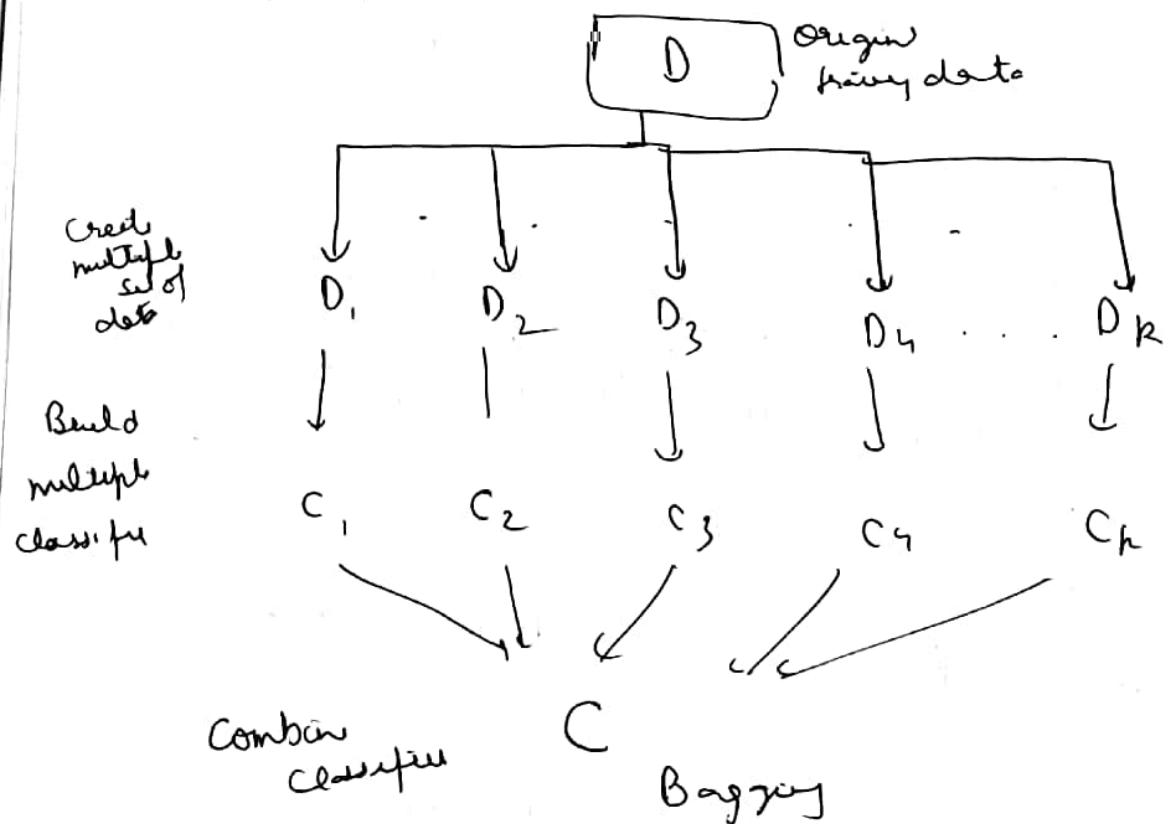
A split uses only one of those m predictors.

→ In Bagging, each decision tree in the ensemble is trained on random bootstrap ~~under~~ sample from the original dataset. However, they considered all features while doing splits.

→ In R.F., each decision tree in the ensemble is trained on random subset of features at each node, introducing feature-level randomness.

This helps de correlate trees + reduce overfitting.

LL



Advantages of Random Forest -

- Can be effectively used to estimate missing data.
- Maintains accuracy even when large proportion of data is missing.
- Can balance errors in datasets where classes are imbalanced.
- Can handle huge datasets with large no. of dimensions.

Limitations

- might overfit noisy datasets.
- Each tree is grown to largest extent possible without pruning.
- Random Forest is slow in generating predictions as multiple decision trees are constructed + this process of voting selects the best predictions result.
- R.F. is difficult to interpret as + when compared to a decision-tree model.
- Large memory usage.

Boosting

L23

Boosting improves ML prediction accuracy and performance by converting multiple weak learners into a single strong learning model.

The main difference b/w Boosting & bagging is:-

In bagging; one improves the accuracy by training several of model at once on multiple dataset

In boosting; one trains weak learners one after another Same Data Set Boosting focus on residuals of current model to make progressively better predictions

General boosting mechanism:-

① Boosting algorithm assigns equal weight to each data sample.

It feeds data to the first machine model, called the base algorithm. The base algorithm makes prediction for each sample.

② It assesses the model predictions & increases the weight of samples with a more significant error. It also assigns a weight based on model performance. A model that output excellent prediction will have high amount of influence over final decision.

⑥ passes weighted data to next decision tree

⑦ Algorithm repeats step ② & ③ until training errors are below certain threshold

Type of Boosting

- ① Adaptive Boosting (AdaBoost)
- ② Gradient Boosting (GB)
- ③ Extreme Gradient Boosting (XGBoost)

→ AdaBoost → initially gives same weight to each dataset.

Then model is trained.

It gives more weight to the incorrectly classified items to correct them for next round. Repeat until residual error (diff b/w actual & prediction) is tolerable.

→ GB → build decision tree to correct errors of previous

Build initial model; calculate negative gradient of loss function w.r.t. current model's prediction

Negative gradient for regression is simply the residual (actual target value - current model's prediction)

logistic loss for Binary

Softmax for Multiclass

→ Actual class probability - Predicted class probability

Fit decision tree to the residuals

L28-

↳ use calculated residuals as the new target variable for decision tree.

→ Train the decision tree to predict the residuals

Combine the output of new decision tree with current model's prediction. Use a learning rate (η) to control contribution of new tree.

current-prediction

New-prediction → after η target = fixed as residual

$$\text{Update prediction} = \text{Current-prediction} + \eta \times \text{New prediction}$$

XG Boost

→ optimized + efficient implementation of gradient boosting with additional features like regularization + parallel processing

Key feature -

- Parallelization
- distributed computing
- Cache optimization

Conditional Probability

When happening of an Event E_1 , depends upon the happening of another event E_2 , then the probability of E_1 is called conditional probability denoted by $P(E_1 | E_2)$

$P(E_1 | E_2)$ denote conditional probability for event E_1 , when event E_2 has already happened

If independent E_1 & E_2 then
 $P(E_1) = P(E_1 | E_2)$

$$P(E_1 | E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)} \rightarrow \text{Bayes theorem}$$

$$P(A) = \frac{1}{2}; P(B) = \frac{1}{3} \quad P(A \cap B) = \frac{1}{6}$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{1}{6}}{\frac{1}{3}} = \frac{3}{4}$$

$$P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{2}{4} = \frac{1}{2}$$

$$\begin{aligned}P(A \cup B) &= P(A) + P(B) - P(A \cap B) \\&= \frac{1}{2} + \frac{1}{3} - \frac{1}{6} = \frac{7}{12}\end{aligned}$$

25% student in M

15% student in P

10% student in both M & P.

fail

A student selected randomly

- ① If they failed in P, then find chance of their failure in M.
- ② If they failed in M, then find chance of their failure in P.

$$P(M) = 0.25 = \frac{1}{4}$$

$$P(P) = 0.15 = \frac{15}{100} = \frac{3}{20}$$

$$P(M \cap P) = 0.1 = \frac{1}{10}$$

$$\textcircled{1} \quad P(M|P) = \frac{P(M \cap P)}{P(P)} = \frac{\frac{1}{10}}{\frac{3}{20}} = \frac{2}{3}$$

$$\textcircled{2} \quad P(P|M) = \frac{P(M \cap P)}{P(M)} = \frac{\frac{1}{10}}{\frac{1}{4}} = \frac{2}{5}$$

$$\begin{aligned} P(A) &= P(A \cap B) + P(A \cap B^c) \\ &= P(A|B)P(B) + P(A|B^c)P(B^c) \\ &= P(A|B)P(B) + P(A|B^c)(1 - P(B)) \end{aligned}$$

→ Lab test is 95% effective in detecting disease when it is in fact present

→ Test also yields a false positive result for 1% of healthy person tests.

If 0.5% of population actually has the disease let say a person has a +ve test what is the person probability of having the disease

D → event that tested person has disease

E → event that his test result is +ve

We want to measure $P(D|E)$
person having disease provided result is +ve.

$$P(D|E) = \frac{P(D \cap E)}{P(E)}$$

$$\boxed{\begin{aligned} P(E|D) &= 0.95 \\ P(E|D^c) &= 0.01 \\ P(D) &= 0.005 \end{aligned}}$$

$$P(D \cap E) = P(E|D) \Rightarrow P(D \cap E) = P(E|D) \cdot P(D)$$

$$\frac{P(D \cap E)}{P(D)} = P(E|D) \cdot \frac{P(D)}{P(D)} + P(E|D^c)(1 - P(D))$$

$$P(E) = P(E|D) P(D) + P(E|D^c)(1 - P(D))$$

$$P(D|E) = \frac{P(E|D) \cdot P(D)}{P(E|D) P(D) + P(E|D^c)(1 - P(D))} = \frac{0.95 \cdot 0.005}{0.95 \cdot 0.005 + 0.01 \cdot (1 - 0.005)} = \frac{95}{294} \approx 0.323$$

$$P(D|EE) = \frac{P(EE|D) \cdot P(D)}{P(EE|D) \cdot P(D) + P(EE|D^c) \cdot (1 - P(D))}$$
$$\approx 0.978$$

Hard voting (Majority voter)

Each model votes for single class

Class with most votes, final prediction

→ traditional bagging

Soft Voting (Probability based voting)

Some models can provide probabilistic output
i.e. probability of each class for a given input.

In soft voting, probabilities of each class
across all models are averaged.

Class with highest average probability is chosen
as prediction

Often more accurate than hard voting

If models provide reliable predictions