PREVENTING OVERFITTING IN DECISION TREE

Introduction:

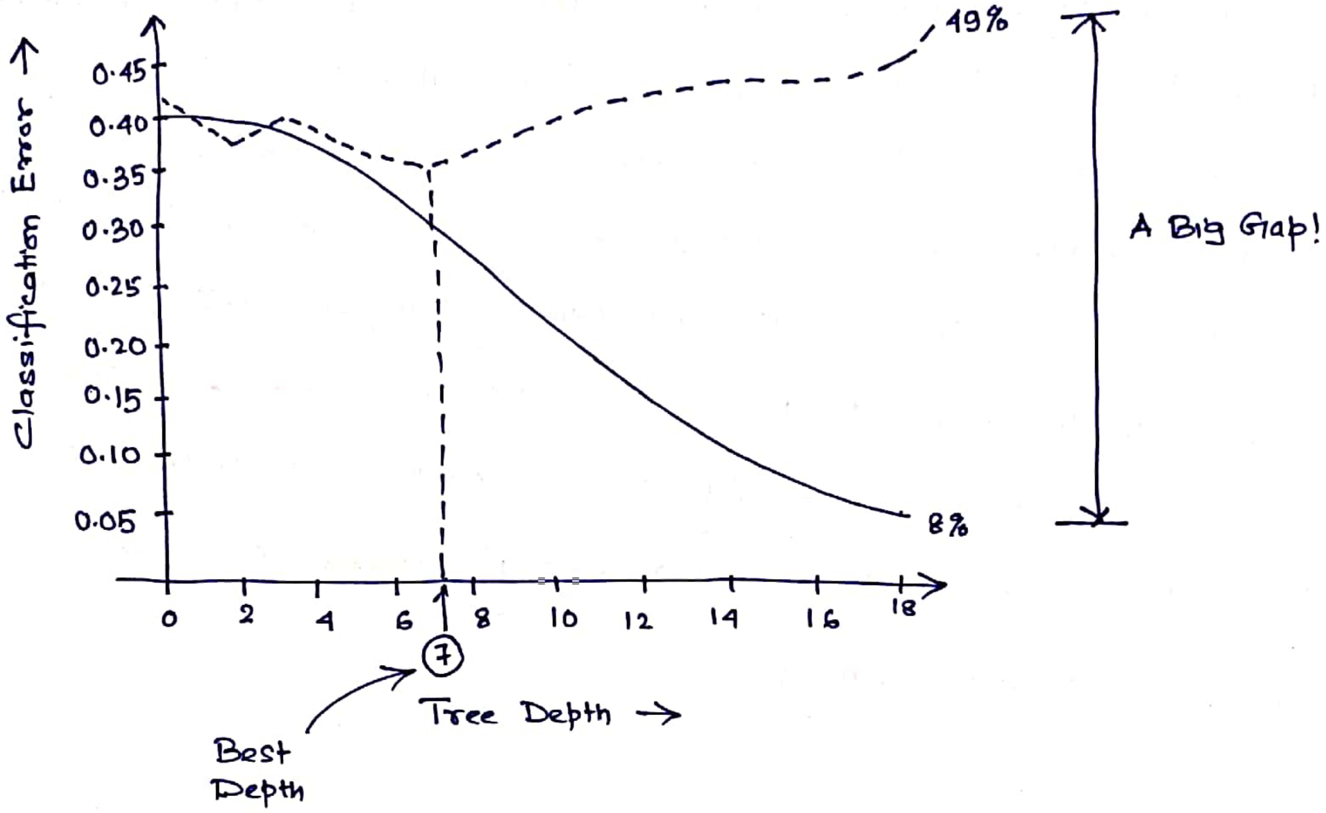
Out of all ML techniques, decision trees are amongst the most prone to overfitting. Therefore, no practical implementation is possible without including approaches that mitigate this challenge. In this module, we will see how we can use the principle of Occum's razor and try to mitigate overfitting by learning simpler trees. We will investigate algorithms that stop the learning process before the decision tree becomes overly complex. We will also investigate a very practical approach that learns a overly complex tree and then simplifies it with pruning.

Overfitting in decision tree:

As we keep increasing the depth of decision tree the decision boundary starts becoming extremely crazzy and the training error tend to reduce to zero. If the training error is extremely less or close to zero then it is definitely a warning signal for overfitting.

Decision tree overfitting - a visual example

Consider a problem of predicting whether a loan application is good or bad and suppose that we experimented fitting a decision trees of different depth and noted down the classification error (training error and validation error) and plotted them as shown in the figure below.



Principle of Occam's Razor:

"Among competing hypotheses, the one with fewest assumptions should be selected."

— William Of Occam, 13th Century.

Occam's Razor for decision trees:

When two trees have similar classification error on the validation set, pick the simpler one.

Example:

Complexity	Train error	Validation Error
Simple	0.23	0.24
PICK Moderate	0.12	0.15
Complex	0.07	0.15
Super Complex	0.00	0.18

same validation error

Overfit

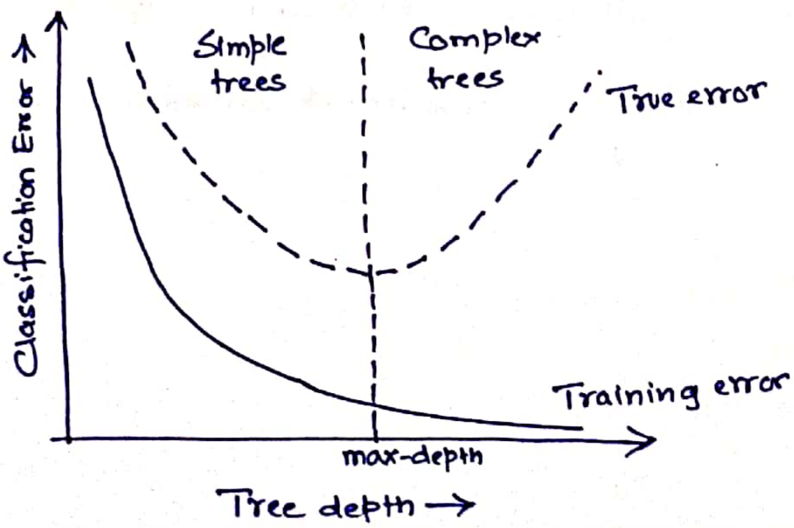
ALGORITHMS to pick Simpler Tree

1. Early stopping rule : Stop learning algorithm before tree become too complex.

2. Pruning : Simplify tree after learning algorithm terminates.

Early Stopping in Learning Decision Tree

- Intuition : Stop growing tree when depth = max-depth (Condition 1)



Challenge: How to pick the max-depth? → Use validation set or cro
validation set.

- Early Stopping Condition 2: Use classification error to limit depth of the tr

Do not consider any split that does not cause a sufficient decrease in classification error.

- Early stopping Condition 3: Minimum node size.

Do not split an intermediate node which contains too few data points.

Challenges in Early Stopping Condition:

(1) For early stopping condition 1 - How do we select the max-depth?

Validation or cross-validation might be a solution, but imagine how many times you will have to return your algorithms and still you might not always reach the desired answer. Also, you might want to grow some parts of the tree more than others. Setting these parameters becomes very complicated.

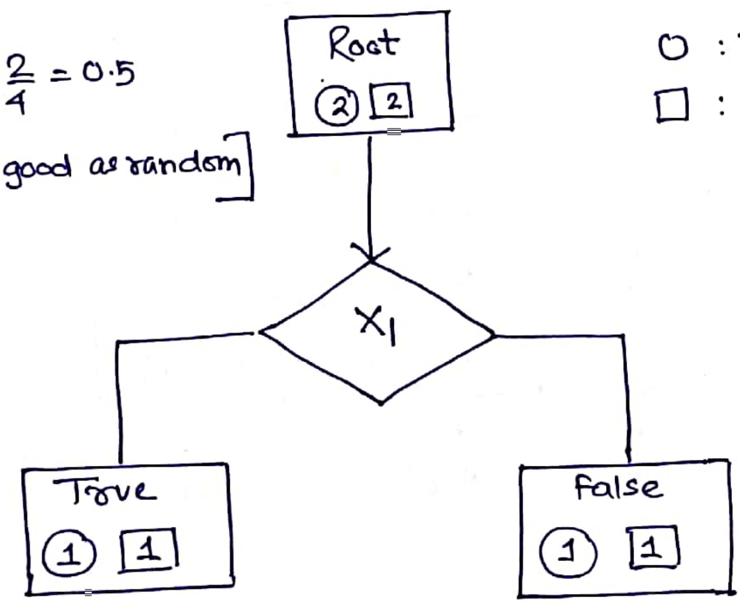
(2) The early stopping condition 2, which says stop if the training error stops decreasing, is the most dangerous one.

counter example:

X_1	X_2	X_1 XOR X_2
false	false	false
false	True	True
True	false	True
True	True	false

Error $= \frac{2}{4} = 0.5$

[Bad; as good as random]



O : True

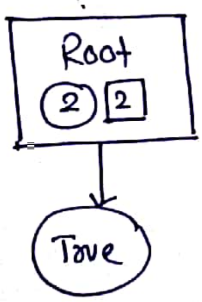
□ : False

Error $= \frac{1}{2} = 0.5$

[Bad; splitting on X_1 does not help]

Note that, the same will be observed while splitting on X_2. Therefore neither features improve training error. So, shall we stop now itself??

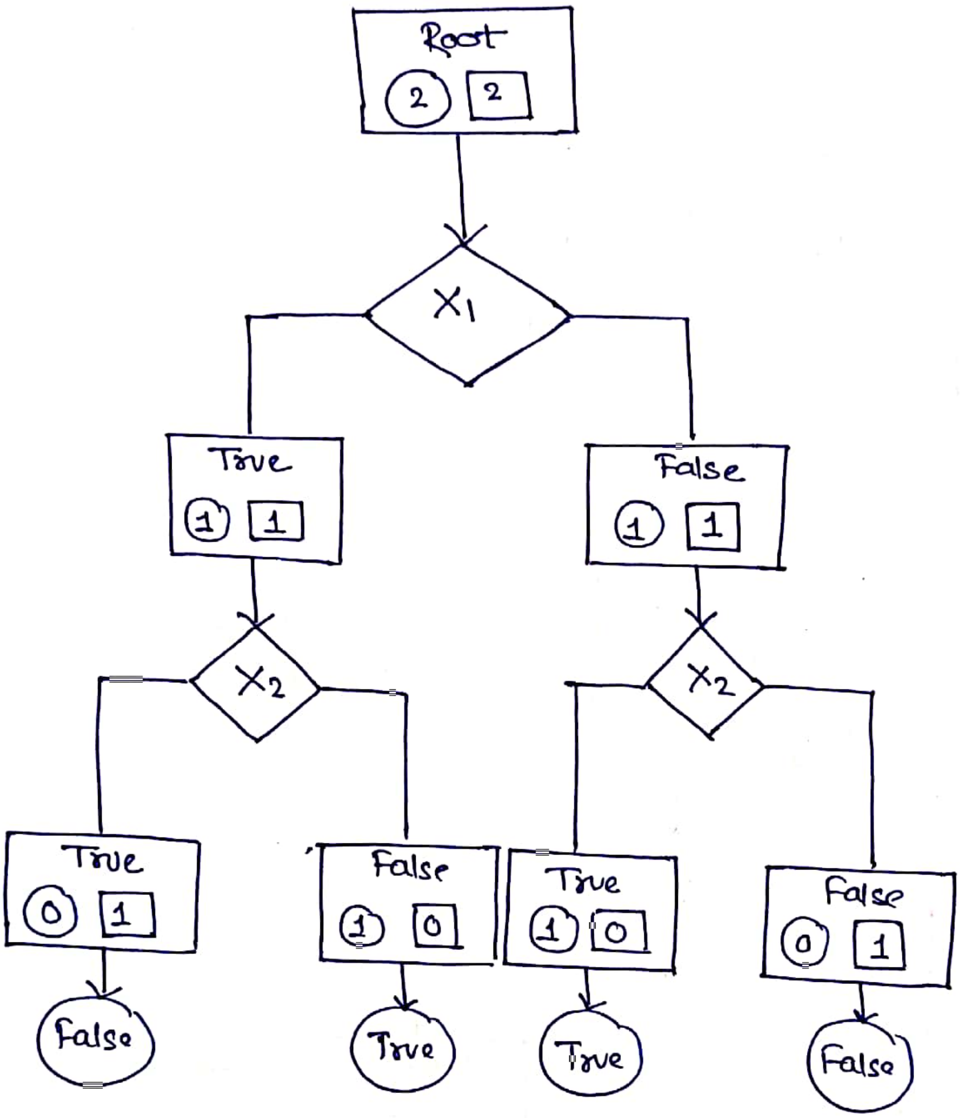
This will make us stop at the root.



Error = 0.5 .

But what will happen if we do not ~~sp~~ stop at stopping conditi

and continue . . .



Error = 0.0

Note that there is a huge gap between training error 0.5 and 0.0.

Pros and Cons of Early Stopping Condition 2:

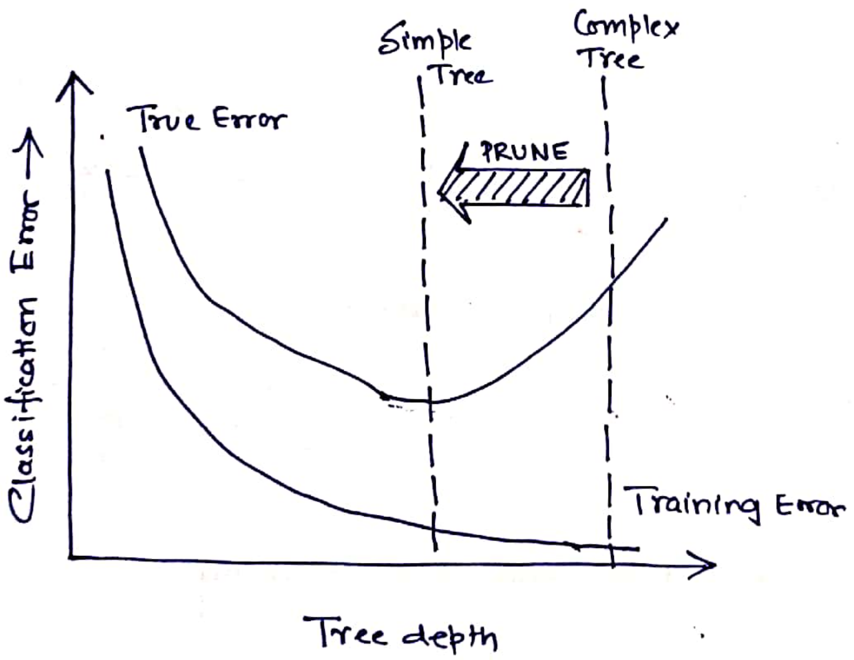
Pros: A reasonable heuristic for early stopping to avoid useless split.

Cons: Too short sighted! We may miss out on "good" splits, which may occur right after "useless splits".

PRUNING

Motivation: (1) Do not stop too early

(2) Simplify after the tree is built



Balance between Simplicity and Predictive Power

A tree becomes more and more complex as

1. its depth increases
2. the number of leaves increases

It is important to balance between simplicity and predictive power.

Cost function:

Total Cost = Classification Error + # of leaf nodes

ie, $c(T) = \text{Error}(T) + L(T)$, where $L(T)$ is the no. of leaf nodes in the tree

in particular,

$$c(T) = \text{Error}(T) + \underset{\substack{\uparrow \\ \text{tuning parameter}}}{\lambda}\, L(T) \quad \text{———} \quad (*)$$

Note that,

If $\lambda = 0$: Standard decision tree learning

If $\lambda = \infty$: infinite penalty $\boxed{\text{Root}} \rightarrow \text{leaf}$ i.e predicting the majority class.

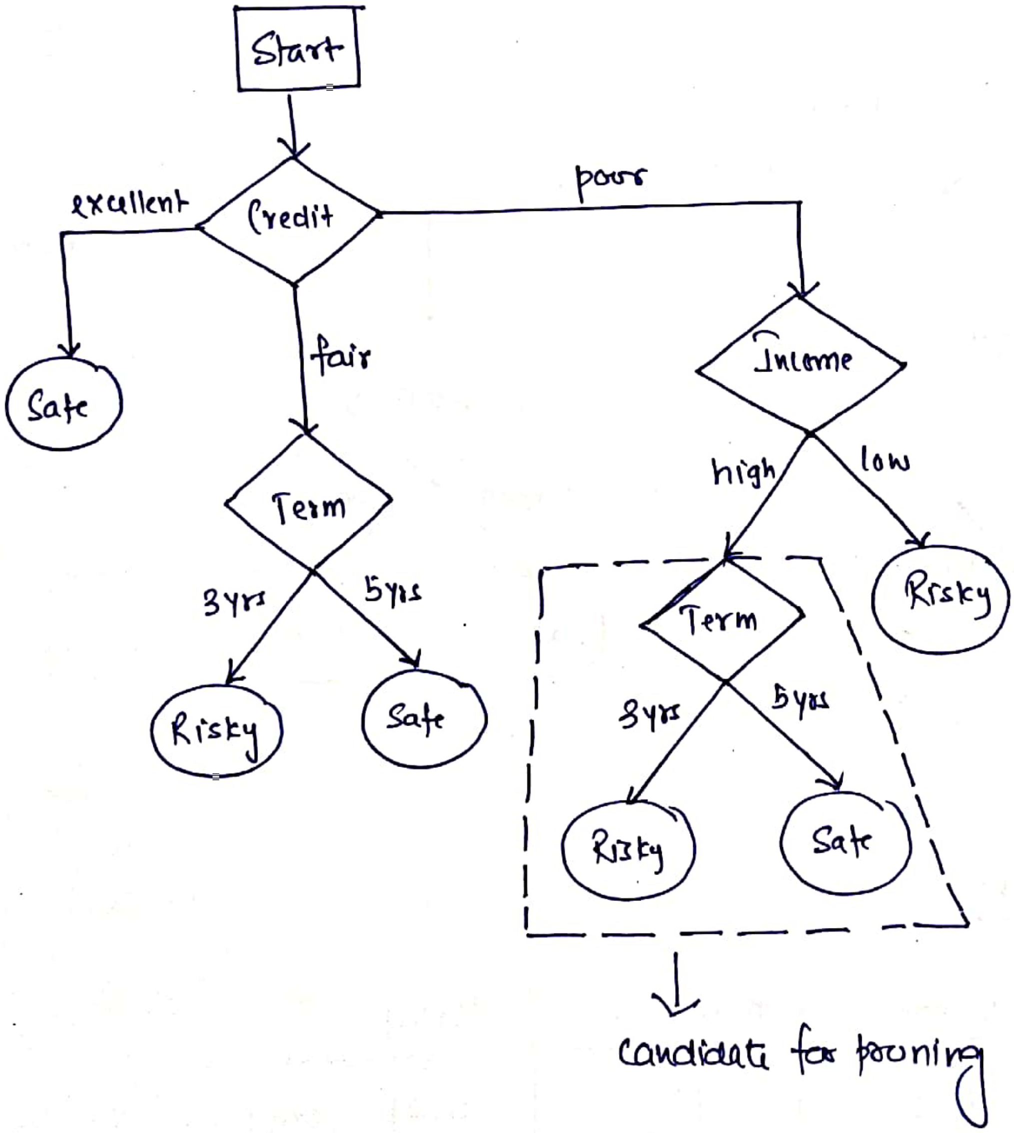
If λ is between : Balance fit and complexity of the tree

Pruning Process

Let us see how the pruning process use the cost function that we have defined and throw away some decisions that are not important.

Steps:

Step 1: Consider a split (usually we start from the bottom)

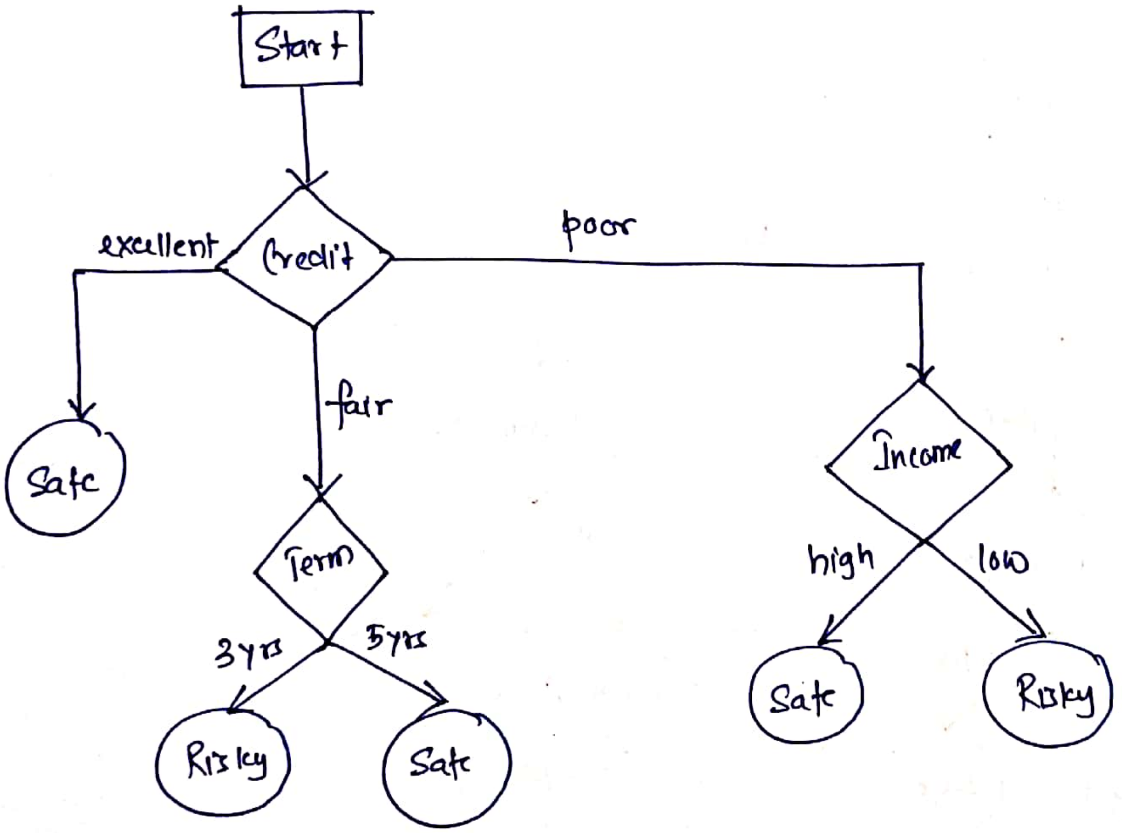


Step 2: Compute total cost C(T) of split

$$C(T) = \text{Error}(T) + \lambda \cdot L(T)$$

Tree	Error	# leaves	Total
T	0.25	6	0.43 (say)

Step 3: Replace split by leaf node and recalculate the cost



Re-calculated Cost:

Tree	Error	# leaves	Total
$T_{smaller}$	0.26	5	0.41 (say)

4: Prune if the total cost is lower.

i.e. if $c(T_{smaller}) \leq c(T)$

Note that in our example, $T_{smaller}$ has worse training error (by a little amount) but has lower overall cost. Therefore ~~the~~ ~~as~~ we must go for the replacement.

Step 5 : Repeat steps 1 to 4 for every split

Visit every splits starting from the bottom and perform the ~~st~~ tasks noted in step 1 to 4.

Decision Tree Pruning Algorithm:

- Start at bottom of the tree T and transverse up, apply prune-split to each decision node M.

- prune-split (T, M):

  1. Compute total cost of tree T using

     $$c(T) = \text{Error}(T) + \lambda \cdot L(T)$$

  2. Let $T_{smaller}$ be the tree after pruning subtree below M.

  3. Compute total cost complexity of $T_{smaller}$

     $$c(T_{smaller}) = \text{Error}(T_{smaller}) + \lambda \cdot L(T_{smaller})$$

4. If $C(T_{smaller}) < C(T)$, prune to $T_{smaller}$.

Additional Notes:

It turns out that as we increase λ from zero in equation (*), branches get pruned from the tree in a nested and predictable fashion, so obtaining the whole sequence of subtrees as a function of λ is easy. We can select a value of λ using a validation set or using cross-validation. We then return to the full data set and obtain the subtree corresponding to λ. This process is summarized in the Algorithm below:

ALGORITHM - 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 no. of observations. (Stopping cond. 3)

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

3. Use K-fold cross-validation to choose λ. That is divide the training observations into K folds. For each $k = 1, 2, \ldots, K$

(a) Repeat step 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 function of λ.

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

4. Return the subtree for step 2 that corresponds to the chosen value of λ.