dim-Reduction; Decision Trees

AW

Lecture Overview

The Idea of Dimsenionality Reduction with PCA

We can use PCA to reduce the dimensions of a problem.

- We know how to de-correlate features by finding good new features (i.e. principal components). Let this map be W
- The percentage of variance (relative energy) that i^{th} principal component carries is the ratio of ith-largest eigenvalue (counting multiplicities) to the total variance (i.e. sum of eigenvalues $= \text{tr}(\text{cov}(D))$ where D is our data).
- Our hope in employing PCA for dimensionality reduction is that:
  - The variance along some principal components is small relatively total variance (i.e. relative energy of these components is small) and therefore must be noise. So let's drop these components. Thus instead of eigenvector $n \times n$ matrix W, we get a new transofmation $W^{'}$ with $n \times d$ matrix where $d < n$. Thus we compress the original data to lower dimensional subspace
  - The variance along the remaining number of principal components provides a reasonable characterization of the complete data set, i.e. we can find decompression map U such that for each data point $\mathbf{x} \in D$ we can claim that $\tilde{\mathbf{x}} \approx UW^{'}(\mathbf{x})$ (note that I replaced denotation $\vec{x}$ by $\mathbf{x}$ for readability.)

Correctness of the Reduction Idea

To show that proposed method is mathematically sound, we must prove that it achieves new goal: reduce n to d, $d < n$, but still have $\min \sum_{i=1}^{m} \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$. More exactly, we must show that choosing d top eigenvstors of $\mathrm{cov}(D)$ to define W transformation solves the optimiztion problem:

$$(S, T) = \arg\min_{U \in \{\mathbb{R}^d \to \mathbb{R}^n\}, W \in \{\mathbb{R}^n \to \mathbb{R}^d\}} \sum_{i=1}^{m} \|\mathbf{x} - UW\mathbf{x}\|^2$$

We need to minimize $\|\mathbf{x} - UW\mathbf{x}\|^2$ for any data point $\mathbf{x}$ as norm is always positive. By Othronormality theorem (slide 14 lecture 2-2) $U = W^T$ when $d = n$ and columns of both are orthonormal (no matter what n and d are). Since columns of $W = U^T$ form basis for $\mathbb{R}^d$ we have $WU = U^T U = 1$. The last part still holds when $U^T \subset W$, so we need

$$\arg\min_{U \in \{\mathbb{R}^d \to \mathbb{R}^n\} : U^T U = 1} \sum_{i=1}^{m} \left\|\mathbf{x}_i - UU^T \mathbf{x}_i\right\|^2$$

$\left\|\mathbf{x} - UU^T\mathbf{x}\right\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x}^T UU^T\mathbf{x} + \mathbf{x}^T UU^T UU^T\mathbf{x} = \|\mathbf{x}\|^2 - \mathbf{x}^T UU^T\mathbf{x} = \|\mathbf{x}\|^2 - \mathrm{tr}(U^T\mathbf{x}\mathbf{x}^T U)$, so we need $\max$ of negative term.
Since trace (as an operator $\mathrm{tr} : \mathbb{R}^n \times \ldots \mathbb{R}^n \to \mathbb{R}$) is linear we can write

$$\arg\min_{U \in \{\mathbb{R}^d \to \mathbb{R}^n\} : U^T U = 1} \sum_{i=1}^{m} \left\|\mathbf{x}_i - UU^T \mathbf{x}_i\right\|^2 = \arg\max_{U \in \{\mathbb{R}^d \to \mathbb{R}^n\} : U^T U = 1} \mathrm{tr}\left(U^T \left(\sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T\right) U\right)$$

Correctness of the Reduction Idea

$$\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\min}\sum_{i=1}^{m}\left\|\mathbf{x}_i-UU^T\mathbf{x}_i\right\|^2=\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\max}\operatorname{tr}\left(U^T\left(\sum_{i=1}^{m}\mathbf{x}_i\mathbf{x}_i^T\right)U\right)$$

By column-row expansion of matrix multiplication we have

$$AB=(\mathbf{col}_1,\ldots,\mathbf{col}_n)\begin{pmatrix}\text{row}_1\\ \vdots\\ \text{row}_n\end{pmatrix}=\mathbf{col}_1\text{row}_1+\cdots+\mathbf{col}_n\text{row}_n$$

, so using it we get

$$\textstyle\sum_{i=1}^{m}\mathbf{x}_i\mathbf{x}_i^T=DD^T=(D^TD)^T=D^TD=(n-1)\text{cov}(D)$$

and as consequence

$$\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\min}\sum_{i=1}^{m}\left\|\mathbf{x}_i-UU^T\mathbf{x}_i\right\|^2=\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\max}\operatorname{tr}\left(U^T\cdot\text{cov}(D)\cdot U\right)$$

Correctness of the Reduction Idea

$$\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\min}\sum_{i=1}^{m}\left\|\mathbf{x}_i-UU^T\mathbf{x}_i\right\|^2=\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\max}\operatorname{tr}\left(U^T\left(\sum_{i=1}^{m}\mathbf{x}_i\mathbf{x}_i^T\right)U\right)$$

Since

$$\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\max}\operatorname{tr}\left(U^T\left(\sum_{i=1}^{m}\mathbf{x}_i\mathbf{x}_i^T\right)U\right)=\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\max}\operatorname{tr}\left(U^T\cdot\operatorname{cov}(D)\cdot U\right)$$

the optimization problem becomes to find

$$S=\underset{U\in\left\{\mathbb{R}^d\to\mathbb{R}^n\right\}:U^TU=1}{\arg\max}\operatorname{tr}\left(U^T\cdot\operatorname{cov}(D)\cdot U\right)$$

Theorem *Dimensionality reduction*
Let $\vec{x}_1,\ldots,\vec{x}_m$ be arbitrary vectors in $\mathbb{R}^n$, let $D=(\vec{x}_1,\ldots,\vec{x}_m)$ be a matrix formed by these vectors. Let also $\vec{u}_1,\ldots,\vec{u}_d$ be d eigenvectors of the matrix $\operatorname{cov}(D)$ corresponding to the largest d eigenvalues of $\operatorname{cov}(D)$. Then, the solution to the PCA optimization problem above is to set S to be the matrix whose columns are $\vec{u}_1,\ldots,\vec{u}_d$ and to set $W=S^T$.

PCA-based Dimensionality Reduction in **R**

Example:

In PCA-dimensionality reduction we want to keep in represntation of Iris data 97% of relative energy.

First run PCA-rotation on Iris data as in last classThen continue with this code

```
#----------------------# dimensionality reduction
varpct < −spectrum/totvar; varpct # prct of variance by component
plot(varpct, xlab = "Principal Component",
   ylab = "Proportion of Variance Explained",
   type = "b") # plots pricipal component number vs variance %
for (i in 1 : d){
   if (varpct[i] < 0.03) break
   } # i is the number of first component that carries less that 3% of variance
Prcvec < −prc$rot[, 1 : (i − 1)] # keep all components that carry > than 3% of
variance
newdat < −as.matrix(iris[1 : d])% ∗ %as.matrix(Prcvec) # new data
```

Lecture Overview

Framework

In statistical setting, the learning algorithm has access to:

- Domain set: An arbitrary set X of instances (data points) that we may wish to label.
  - Usually data points are represented by a vector of features
- Label set: for now we assume that a label set Y is a two-element set. Usually either $Y = \{0, 1\}$ or $Y = \{-1, +1\}$
- Training data: $S = \{(x_1; y_1), \ldots (x_m; y_m)\}$ is a finite sequence of pairs in $X \times Y$, that is, a sequence of labeled data points.

  Training data is the input to which classifier has access

The algorithm must output:

- A classifier $h : X \to Y$. This function is also called a predictor, a prediction rule or a hypothesis. The classifier can be used to predict the label of new data points; $A(S)$ denote the hypothesis that algorithm, A, returns upon receiving the training sequence S.

Assumptions

We assume a simple data-generation model:

- the instances are generated by some probability distribution over a domain set X. Denote this probability distribution over X by $\mathcal{D}$.
- $\mathcal{D}$ could be any arbitrary probability distribution. We do not assume that the learning algorithm knows anything about this distribution.
- there is some correct labeling function, $f : X \to Y$ and $y_i = f(x_i)$ for all i.
- each pair in the training data S is generated by first sampling a point x_i according to $\mathcal{D}$ and then labeling it by f
- the labeling function is unknown to the learner
- the goal of the learning algorithm it to learn $h = f$

Measures of success:

- For now we define the error of a classifier h to be the probability to draw a random instance x, according to the distribution $\mathcal{D}$, such that $h(x)$ does not equal $f(x)$.

Generalization Error and Empirical Risk Minimiation

Generalization error of a classifier $h :\rightarrow \{0,1\}$ with respect to a data probability distribution $\mathcal{D}$ on data data domain X and labeling function $f : X \rightarrow \{0,1\}$ is

$$L_{\mathcal{D},f}(h) \stackrel{def}{=} \Pr_{x\sim\mathcal{D}}(h(x) \neq f(x)) \stackrel{def}{=} \mathcal{D}(\{x : h(x) \neq f(x)\})$$

The problem is that the true error is not directly available to the learner. A useful notion of error that can be calculated by the learner is the training error i.e. the error the classifier incurs over the training sample $S = (x_1, y_1), \ldots, (x_m, y_m)$:

$$L_S(h) \stackrel{def}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}$$

Training error is also know as empirical risk of h. Natural goal for learning algorithm is to come up with a predictor h that minimizes empirical risk $L_S(h)$. This type of learning is known as Empirical Risk Minimization (ERM).

Restricting Class of Hypothesis

If we allow ERM learner to use any hypothesis then in ERM it'll choose one that perfroms perfectly on training set, and may perform badly on all data (aka overfitting). To avoid it we narrow down class of predictors in which we search in hope to prove that in this class ERM predictor wont overfit.

- Learning algorithm chooses class of predictors $\mathscr{H}$ in advance (before seeing data)
  - Choosing a class $\mathscr{H}$ should ideally be based on some prior knowledge about the problem to be learned.
  - Restricting predictors to $\mathscr{H}$ means biasing learner toward certain class of solutions this is known as *inductive bias*
- Given class $\mathscr{H}$, and a training sample S, the ERM$_{\mathscr{H}}$ learner uses the ERM rule to choose a predictor $h \in \mathscr{H}$ with the lowest possible error over S, i.e. it finds a solution to

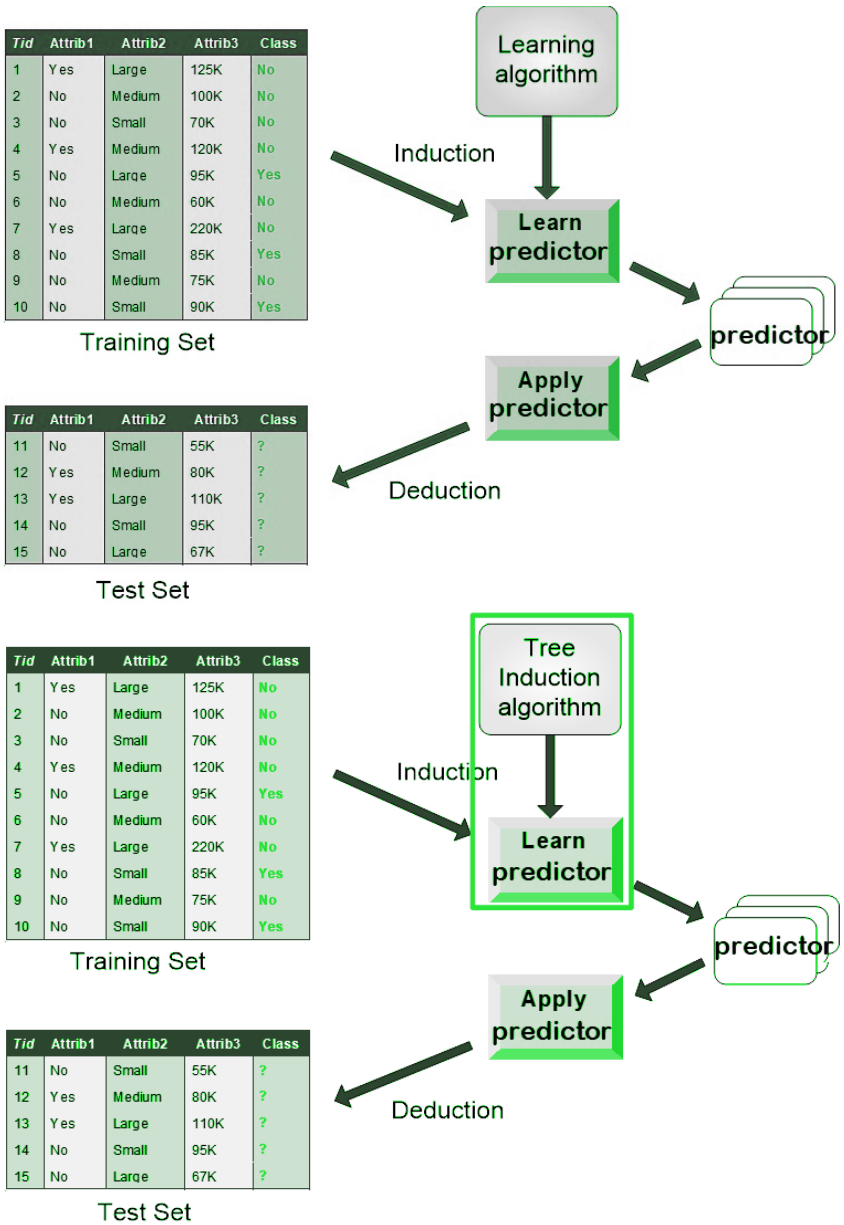
$$\text{ERM}_{\mathscr{H}}(S) = \underset{h \in \mathscr{H}}{\arg\min}\, L_S(h)$$

Lecture Overview

Hypothesis Class and Learning Process

- Learning algorithm uses training data to obtain a predictor and testing data to evaluate quality of the predictor
- For a data set D that is n-dimensional space we choose the hypothesis class to be n-dimesional rectangles
- Rectangless are obtained by recursive dividing of existing rectangles
- At each step a rectangle for splitting is chosen. It is then divided into 2 by 'drawing' an axis-parallel plane that divides it in two.
  - Originally the plane was parallel to one of the axeses. In today's implementations it can be any plane

This learner is known as Inductive



How the DT predictors work

Descritpion of a simplest DT predictor - binary tree classifier:

- At each internal node an instance is tested for the value of a feature assigned to this node
- A node tests an assigned feature X_i for being either $=$ or $\leq$ than a given X_i-domain value v_i.
- If an instance has feature value $X_i = x_i$ that makes the internal node test `False` then the instance is forwarded to the left child of the node, if the test results in `True` then the instance is forwarded to the right child of the node
  - this is "drawing" a plane parallel to axis X_i at v_i step
- The process is repeated until an instance reaches the leaf of the tree
- Each leaf has an assigned class. The instance is assigned a class of the leaf it reached

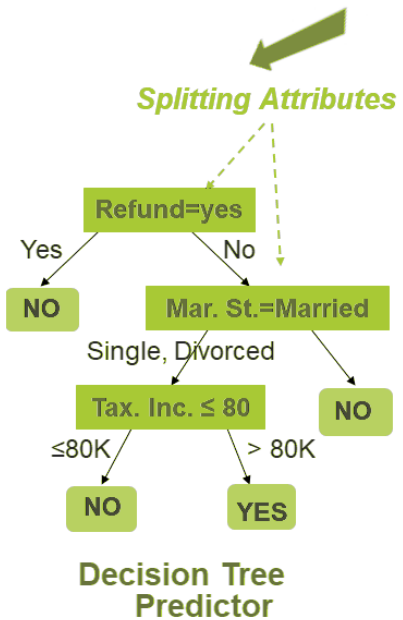
Decision Tree Predictors

- The result of learning algorithm is a Decision Tree (DT) predictor that given an instance associates to it a class.
- For a given traning data S there can be many possible DT predictors.
- Learning process must return a DT predictor that minimizes $\text{ERM}_{\text{DT}}(S)$
- But there may be more than 1 predictor that minimizes $\text{ERM}_{\text{DT}}(S)$

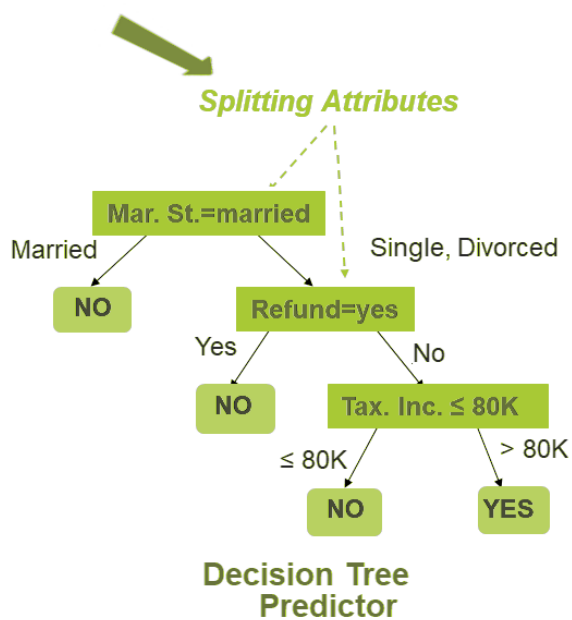
Example:

Training Data

Tid	Refund	Marital Status	Taxable Income	Class: Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



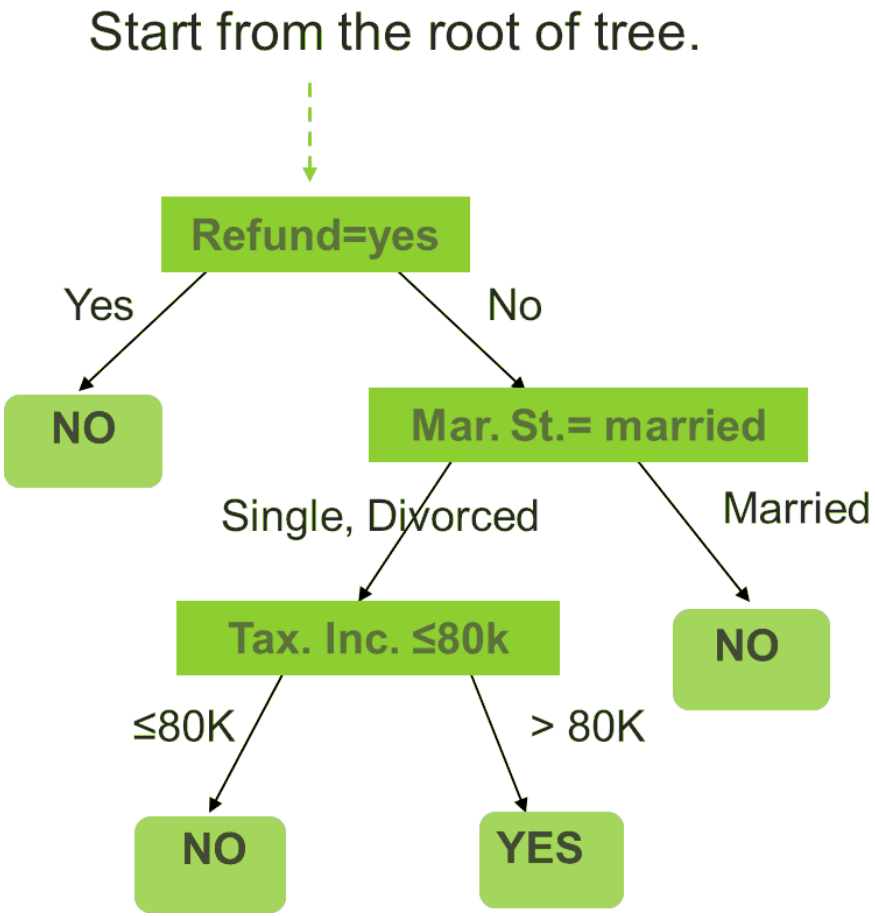
Decision Tree Predictor



Decision Tree Predictor

Lecture Overview

Example: Cheating on Taxe Returns



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Figure: Classification of a New Instance

Example: Cheating on Taxe Returns

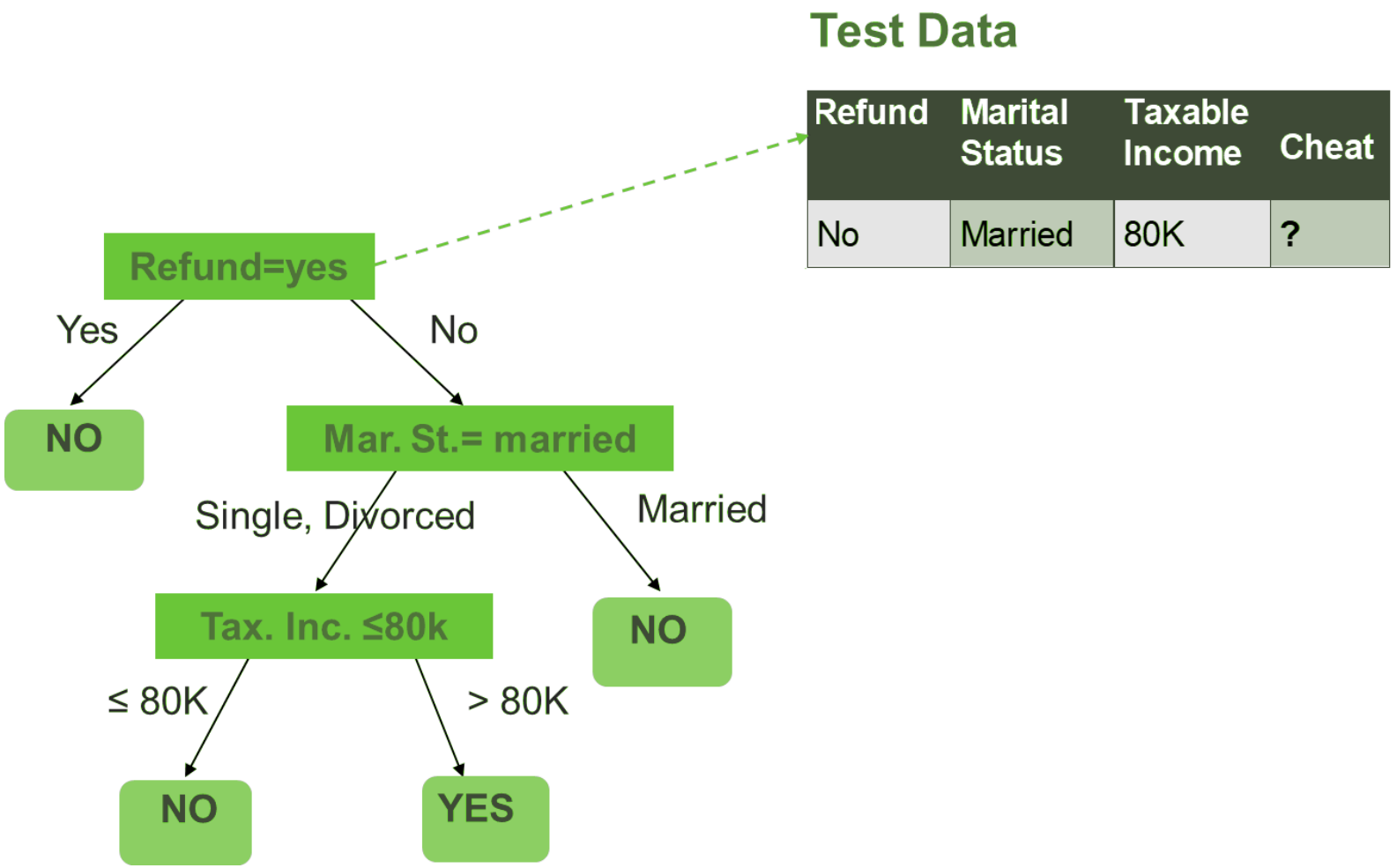


Figure: Classification of a New Instance

Example: Cheating on Taxe Returns

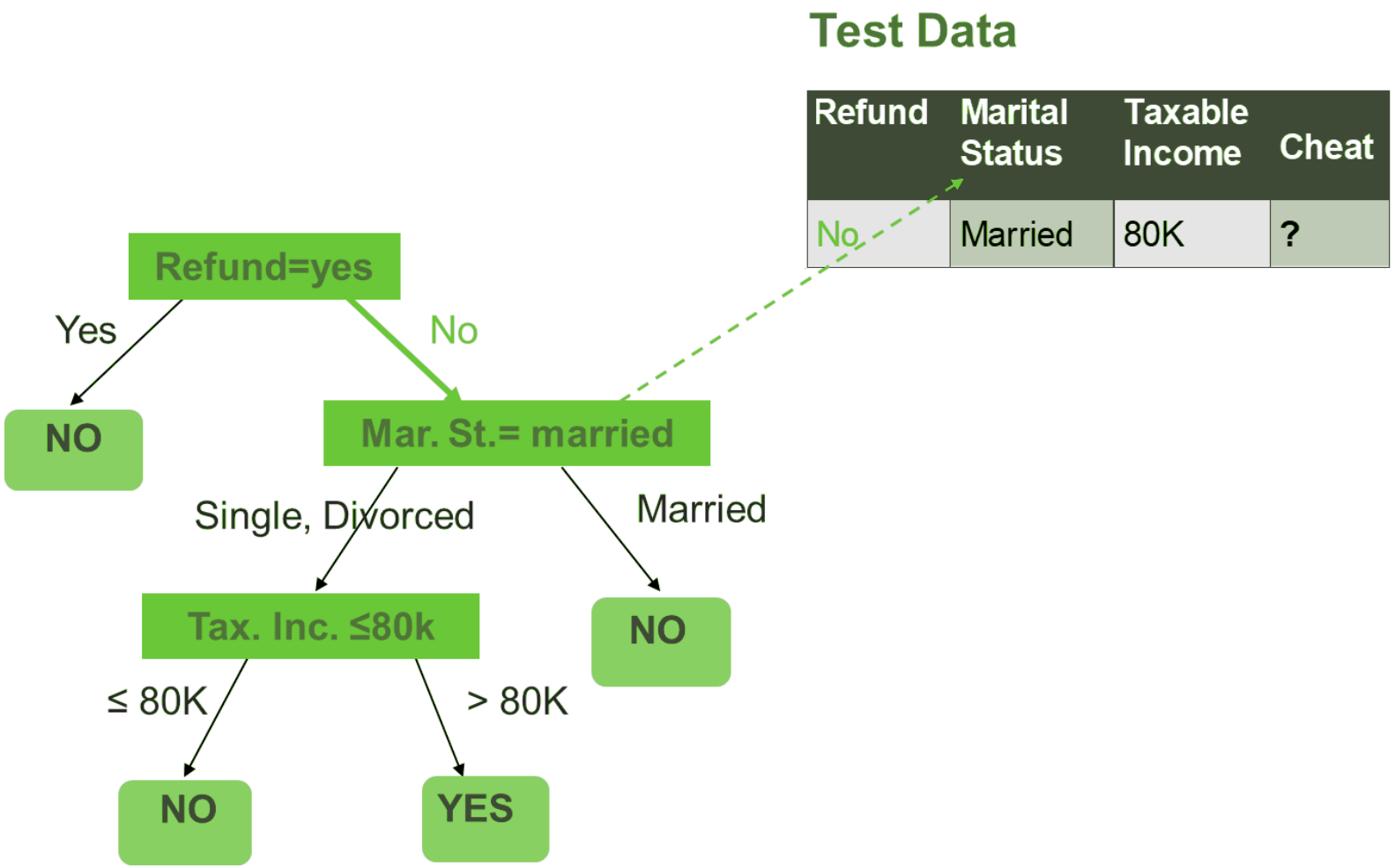


Figure: Classification of a New Instance

Example: Cheating on Taxe Returns

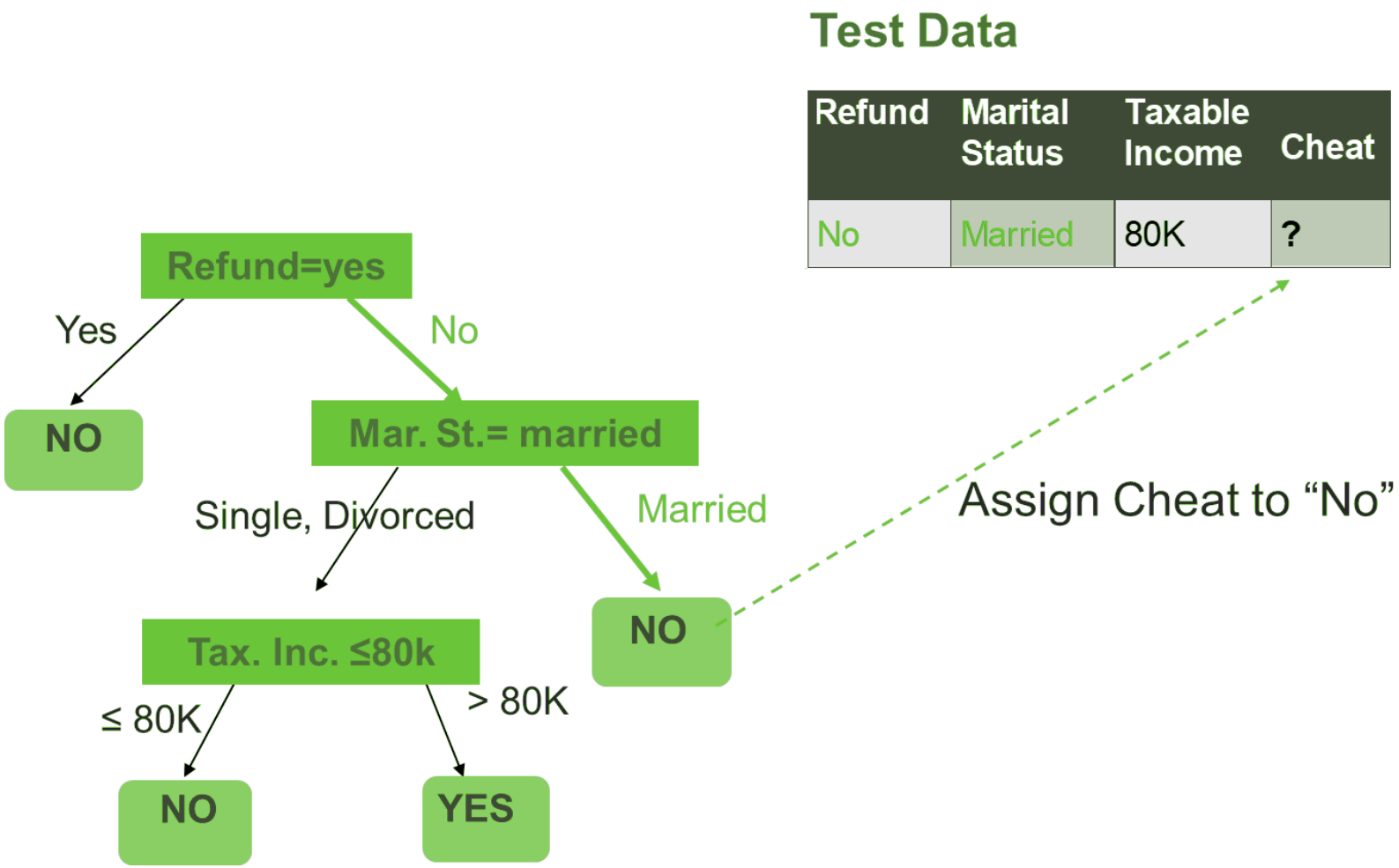


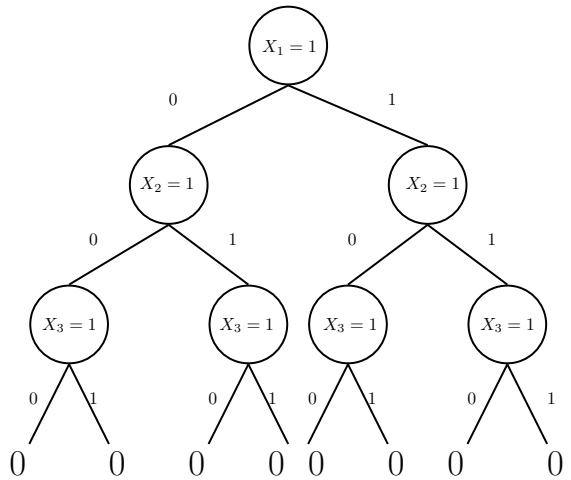
Figure: Classification of a New Instance

Expressive power of DT predictors

What kind of hypothesis class is DT?

- The figure shows that we can get any boolean function $f : \{0,1\}^n \to \{0,1\}$ on n variables
- If for each feature its domain is finite, then each test in a decision can be ”unfolded” into its values on domain elements (this is $\bigvee$-test, though formula can be huge, it is finite). The DT classifier then computes a characteristic function $f : D_1 \times D_2 \times \cdots \times D_n \to \{0,1\}$.
- note that class of these functions is finite

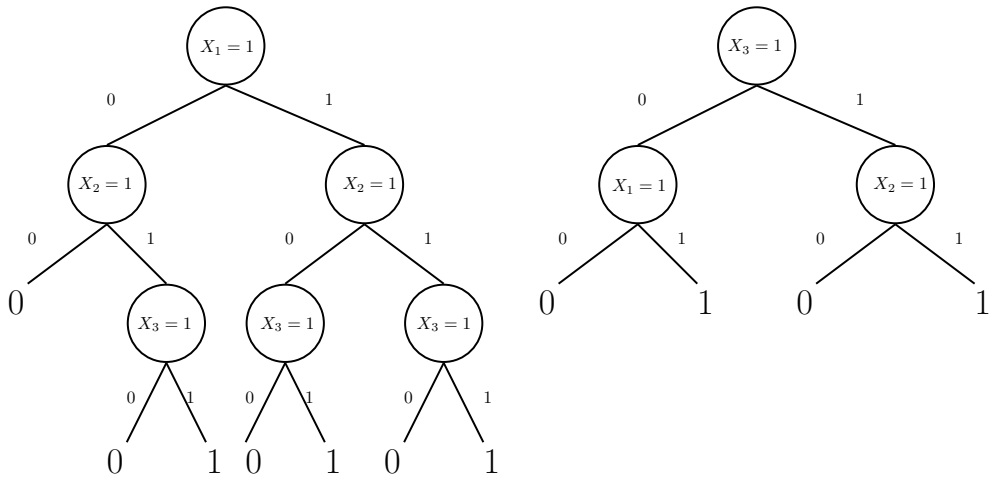
X_1	X_2	X_3	$f(X_1, X_2, X_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



Pivotal Questions

- Given the training data which tree is the best DT?
  - We have the answer - ERM tree. However, if we change the paradigm our answer may change
- ERM tree may not be unique - see example. Which tree shoud we choose if there is more than one ERM tree?

- Intuitively the simpler/smaller the tree the better (less computations on prediction). Can we justify/prove it?
- Suppose we got some DT tree. Learning the simplest/smallest equivalent tree is an $\mathbf{NP}$-complete problem [Hyafil & Rivest 76]



- Given training (incomplete) data, can we learn any ERM tree from it?
  - The answer is yes if domains of all features are finite (we'll show it later). But finiding ERM tree is $\mathbf{NP}$-hard, so we need approximate polynomial algorithm!

Lecture Overview

Learning by Recursive Partitioning - the Idea

1. Take training data set D. Call this data a ”node”, assign to this node type ”leaf”, class of majority of data points and add it to the list of leafs.
2. Take any available leaf node L and remove it from the list of leafs
3. Select the variable X_i and a value $t \in D_{X_i}$ that produces the greatest separation of classes in the training data w.r.t. a chosen mathematical criteria. If no variable provide sufficient separation, go back to step 2
   - The pair (X_i, t) is called a split
   - If $X_i \leq t$ in a data point then send the data to the left set (aka left node); otherwise, send data point to the right set (right node)
   - Assign to both left ad right nodes type ”leaf”, class of majority of data points and add them to the list of leafs
4. Re-assign to original node L type ”internal”
5. Repeat steps 2 to 4 while the least of leafs is non-empty

You get a tree. Can use binary splits or multinode splits.

Learning DT Predictor - Example

Disclaimer: in this example we do not explain how do we choose a split at any given step. We assume that such function already exists and every time we need to choose a split we call this function and get a split in return.

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Node L, leaf, class No

Figure: Initialization of learning Decision Tree

Learning DT Predictor - Example

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

select split on feature "Refund" value "no"

Node L, leaf, class No

Figure: First Split selection

Learning DT Predictor - Example

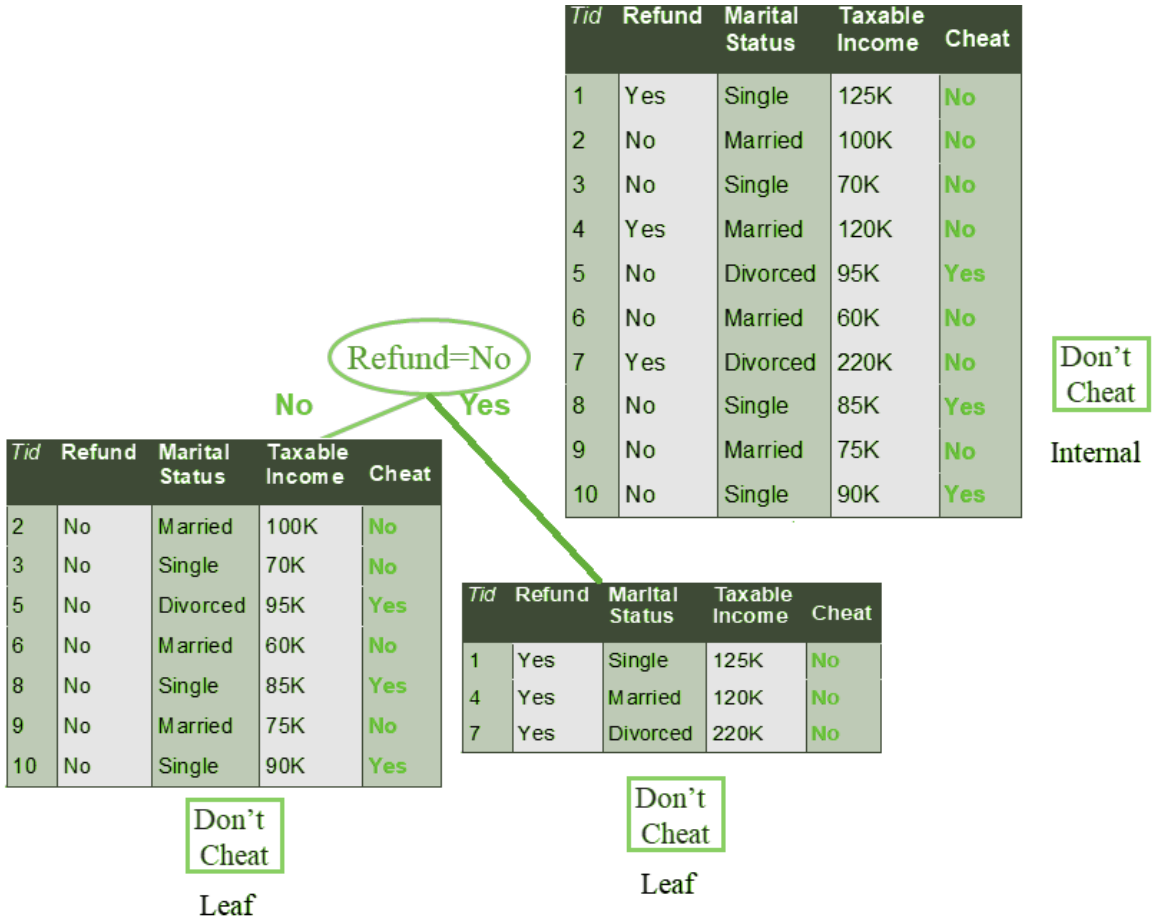


Figure: First Tree Expansion

Learning DT Predictor - Example

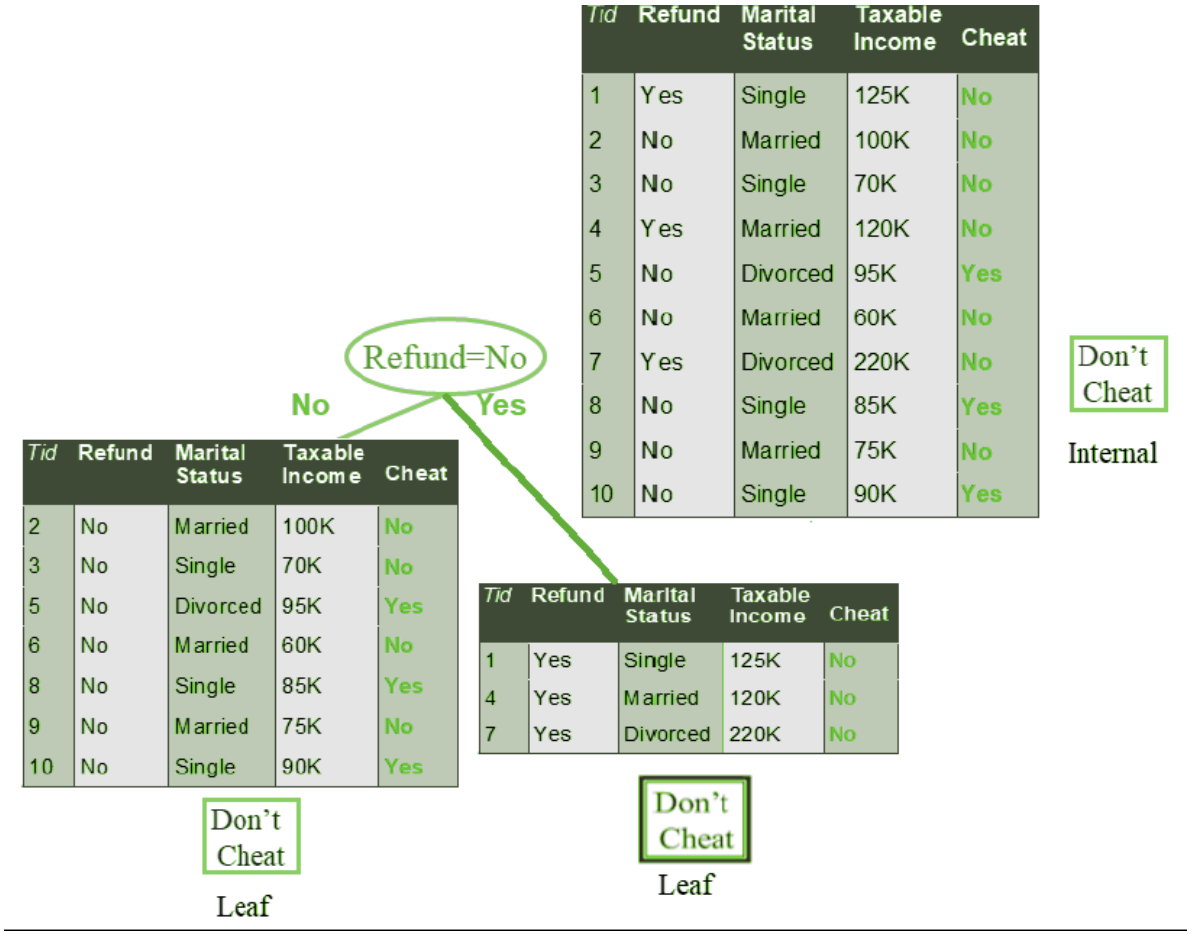


Figure: First Unsplitable Leaf Purge from Leaf-list

Learning DT Predictor - Example

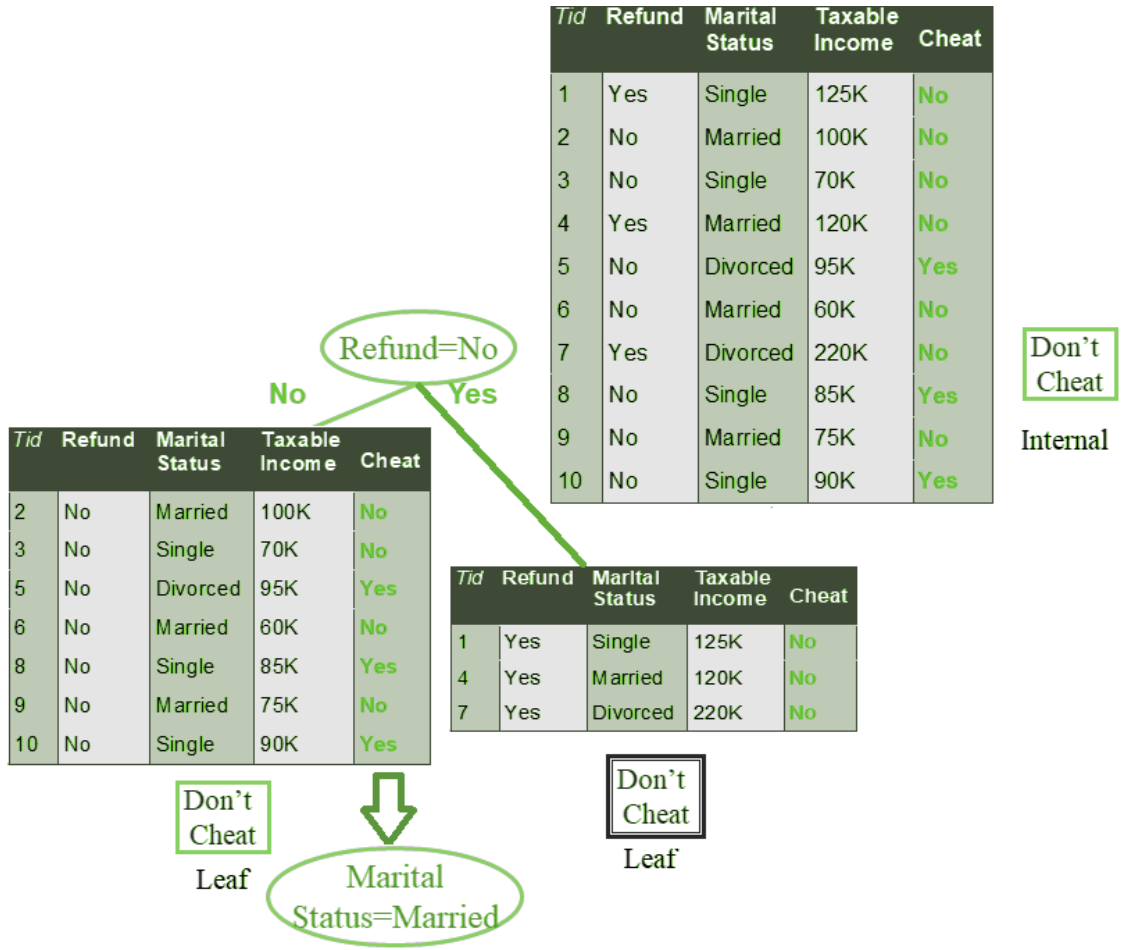


Figure: Second Iteration - Split Selection

Learning DT Predictor - Example

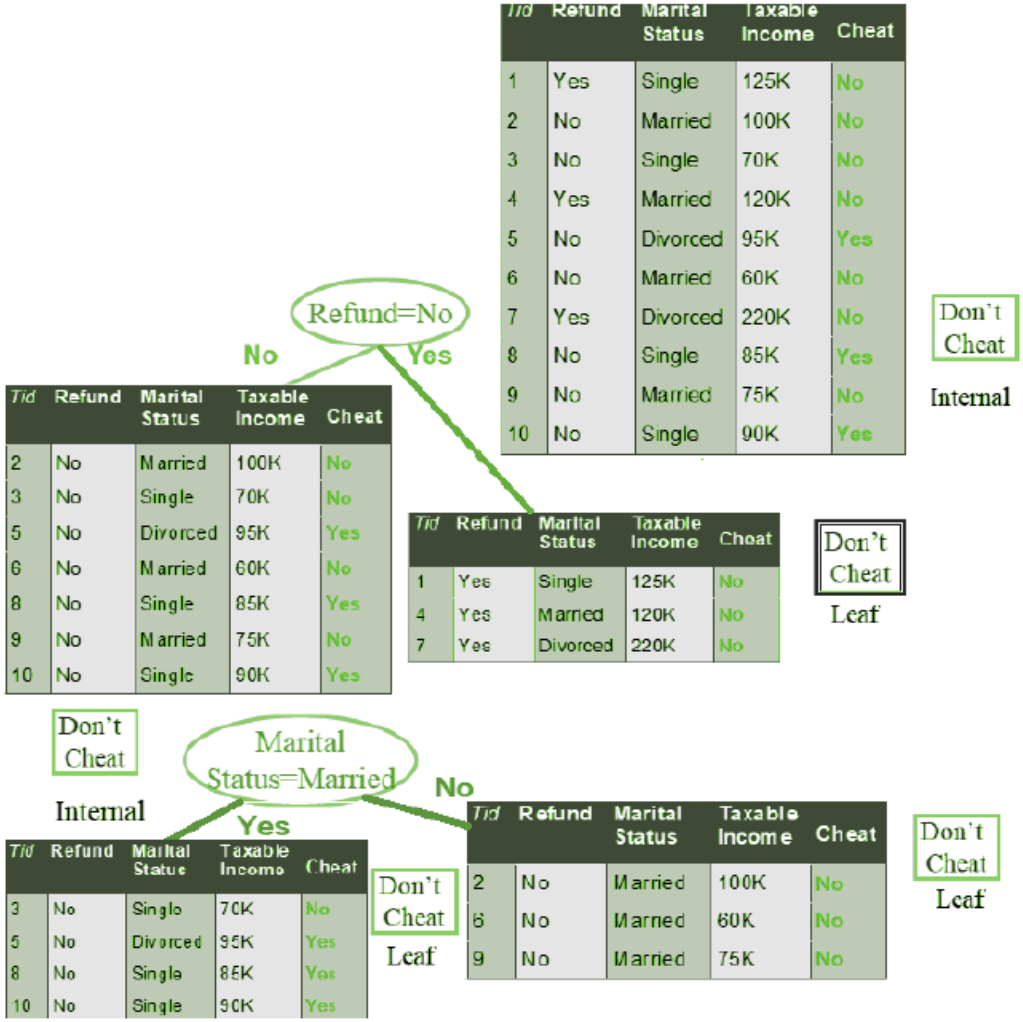


Figure: Second Iteraion - Tree Expansion

Learning DT Predictor - Example

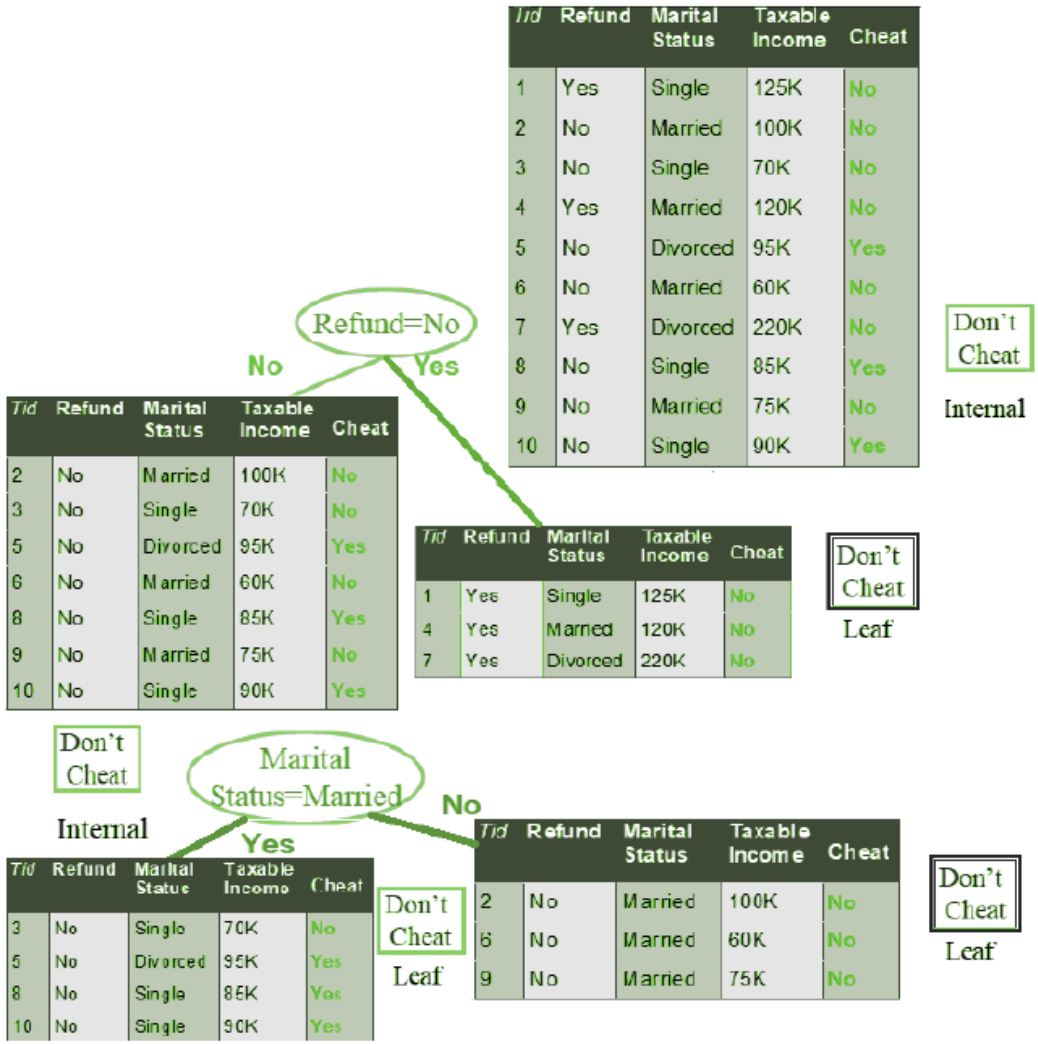


Figure: Second Iteraion - Unsplitable Leaf Purge

Learning DT Predictor - Example

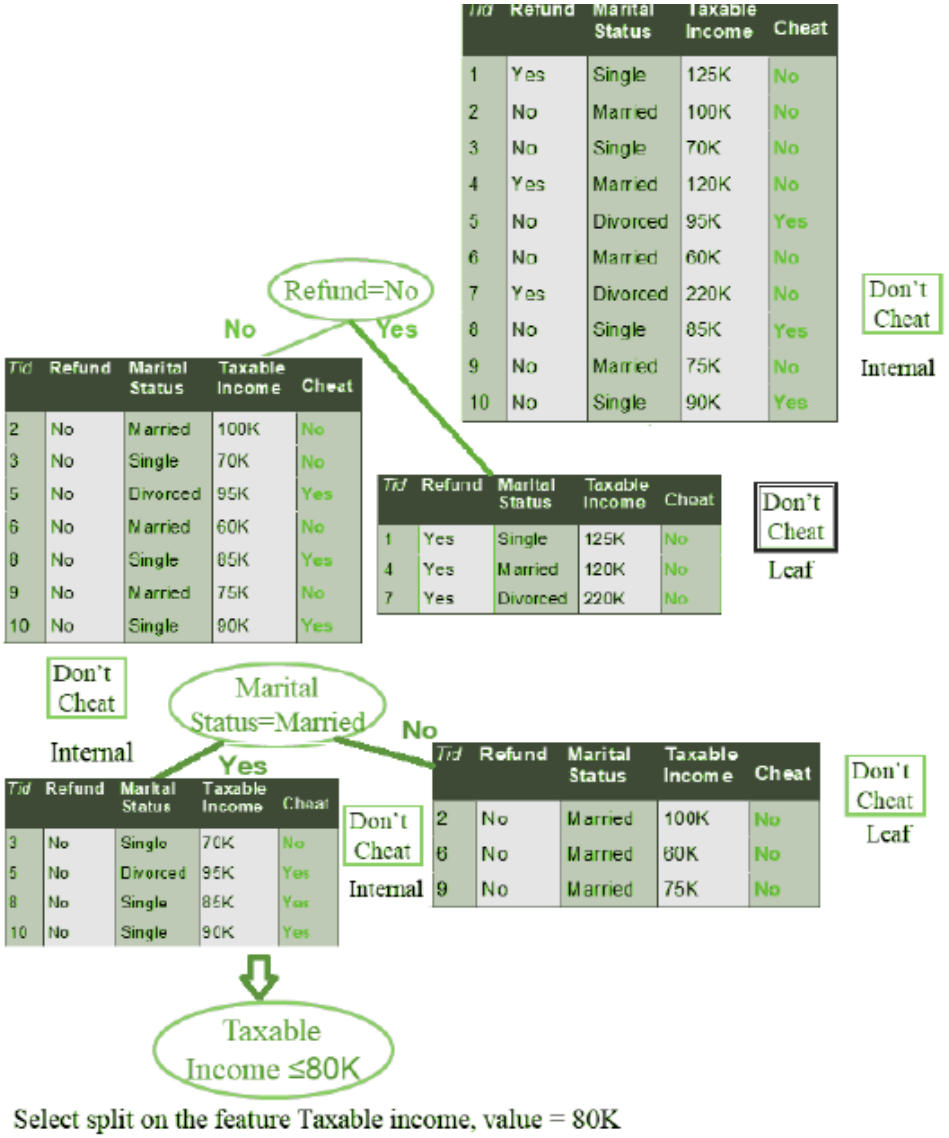


Figure: Thirds Iteraion - Split Selection

Learning DT Predictor - Example

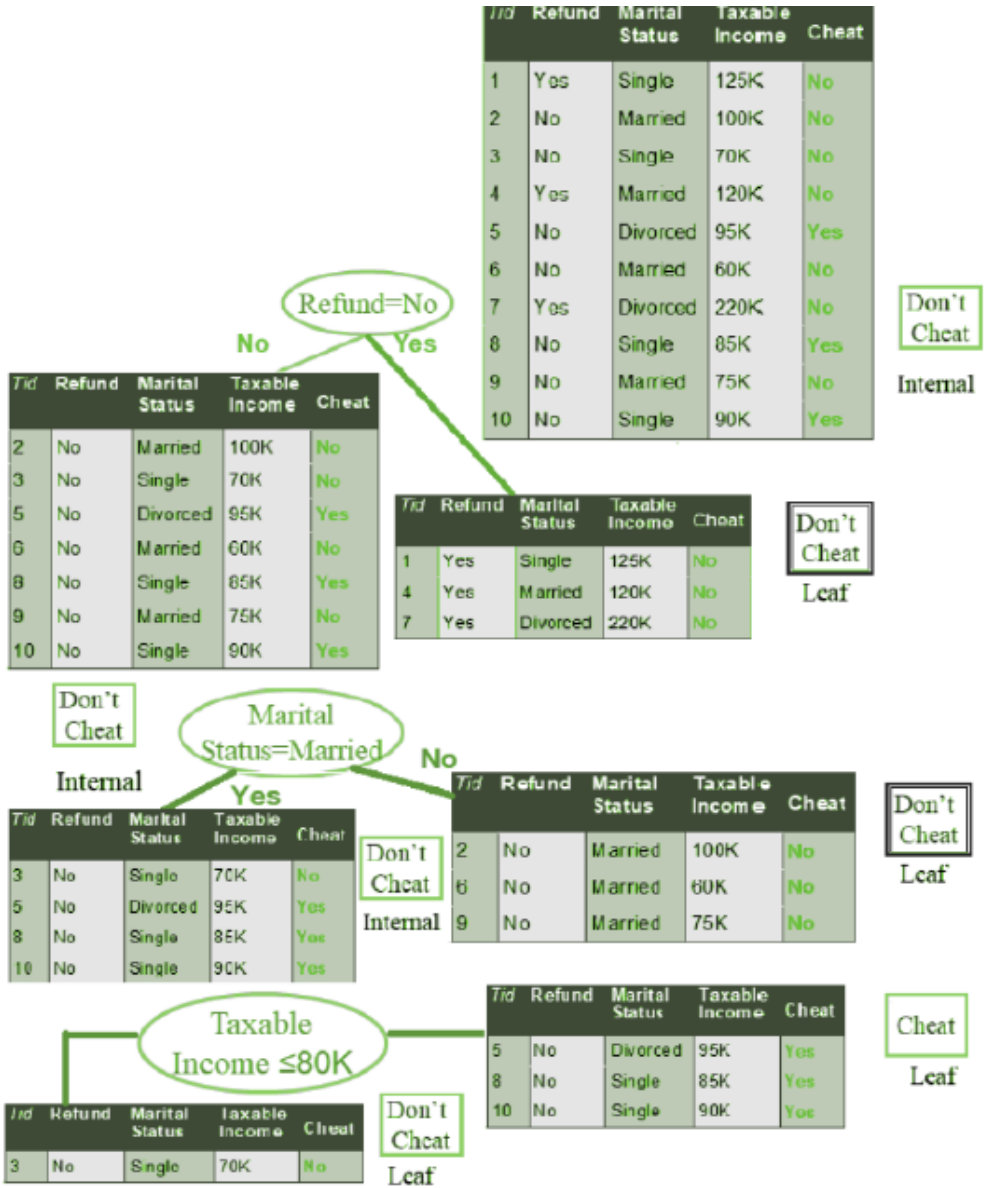


Figure: Thirds Iteraion - Tree Expansion

Learning DT Predictor - Example

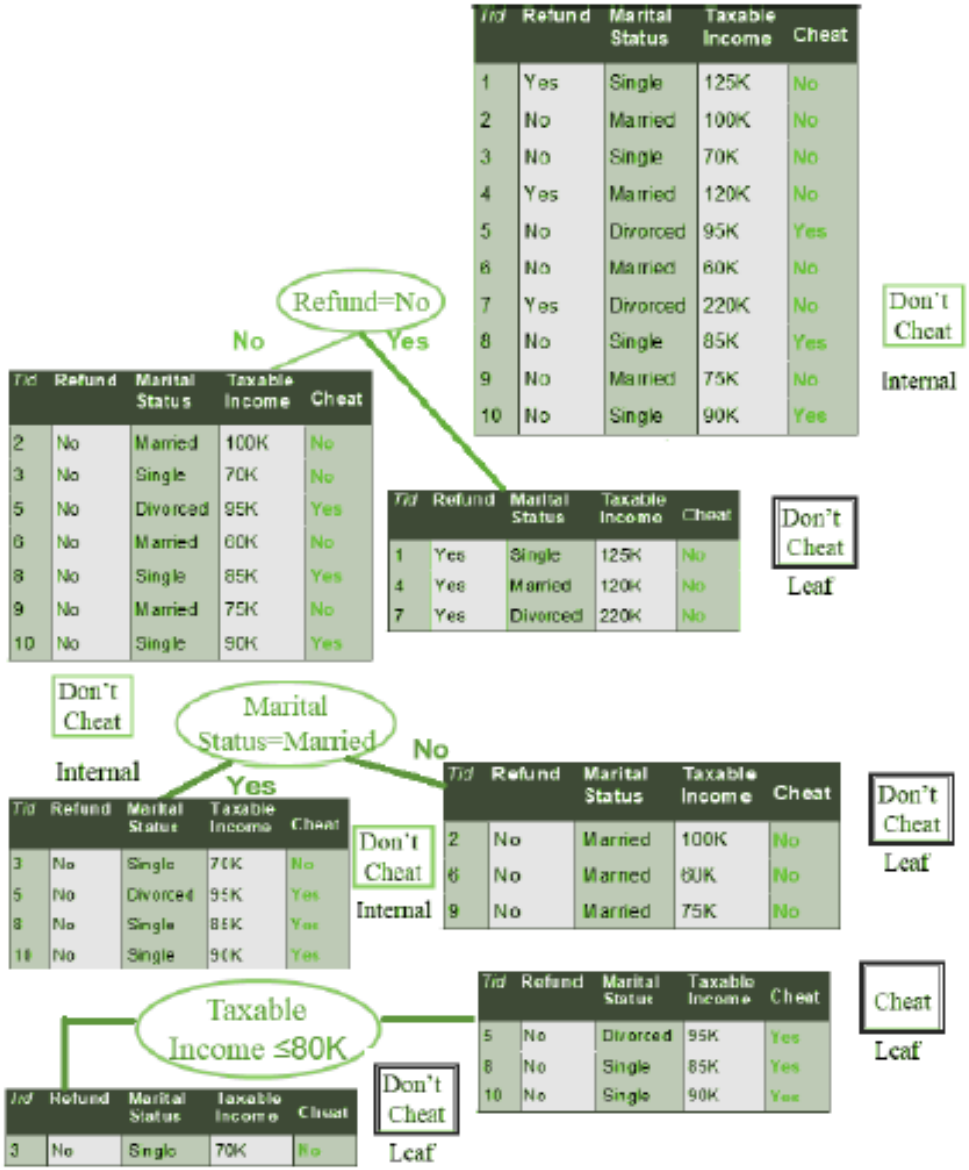


Figure: Second Iteraion - Unsplitable Leaf Purge. Termination

Learning DT Predictor - Example

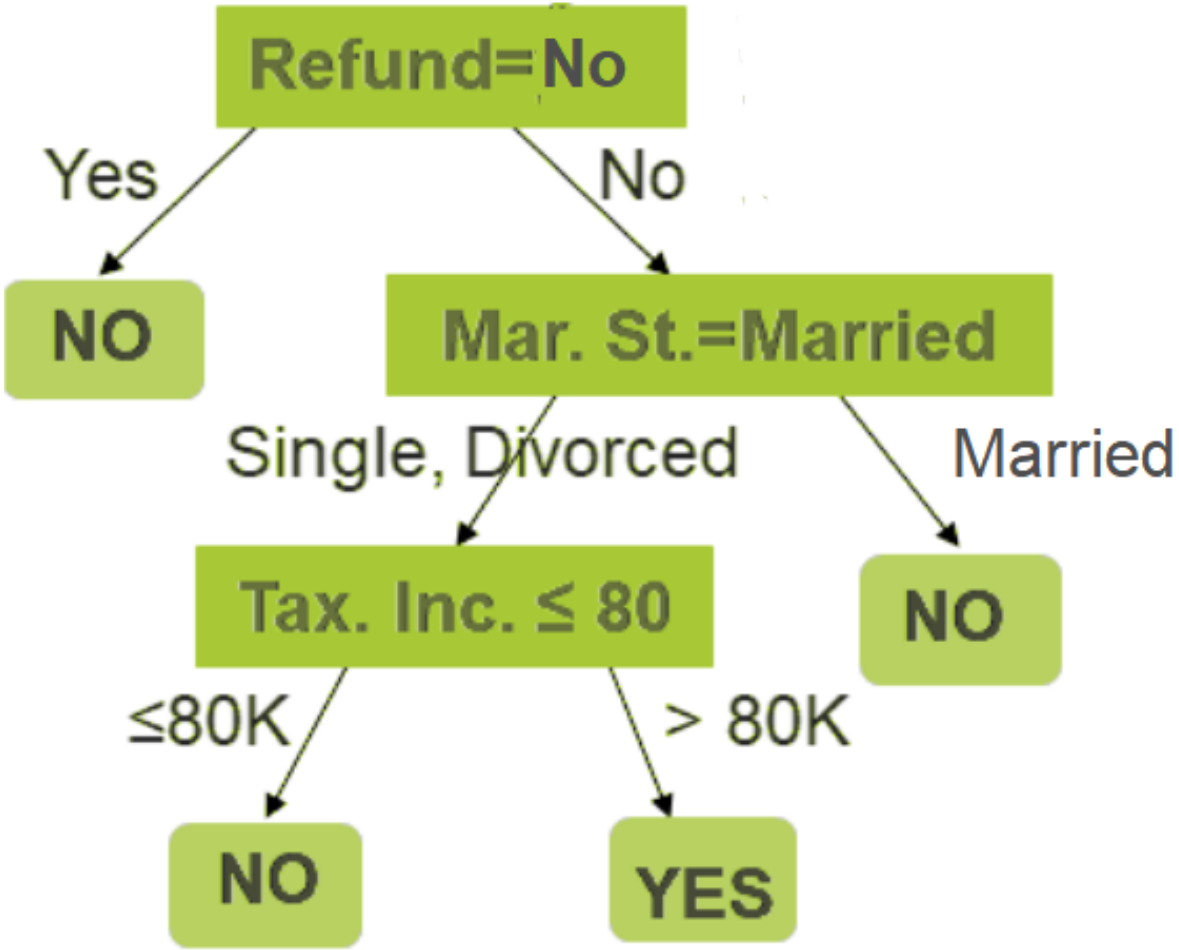


Figure: Learned Decision Tree Predictor with Empiirical Risk=0

Recursive Partitioning is Greedy

- If computing the best split takes polynomial time then the recursive partitioning based algorithm runs in poly-time because at every iteration round we partition the data D into 2 parts, each at least one datapoint smaller than the original data, so the total number of partitioning steps is at most $|D|$
- Step 3 of any recursive partitioning based algorithm involves choosing the split that provides ”greatest separation” of the classes between the nodes *after this step*. The latter means that any recursive partitioning based algorithm is ”greedy”:
  - To find ”greatest separation” of classes we must show that after split the quality of leafs is best achievable *in this step*. So we need
    - Purity-function on for current node evaluating how good the node is w.r.t. assigned class
    - A way to evaluate purity of the expansion nodes produced by potential split to evaluate total purity of the split
    - A way to compute gain of purity obtained by a split
    - Computaton of purity to be ”local”, i.e. without looking ahead and evaluating effects of current split on optimality of final solution
  - The step itself is to maximize gain of the split