Classification and Prediction

Nischal Regmi

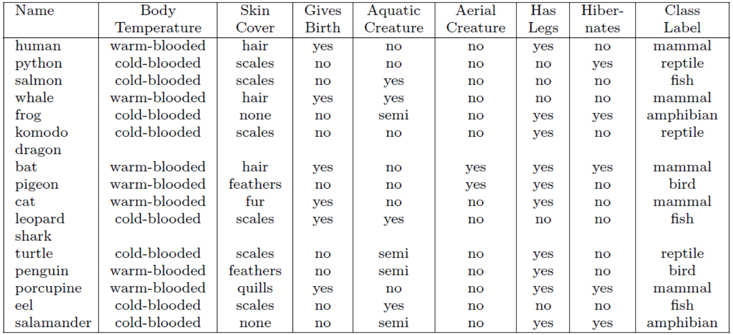
Everest Engineering College

2022

Classification

The Classification Problem

1. We are given a set of D items which are tuples $(\mathbf{x}, y)$ where $\mathbf{x}$ is called the attribute set and y is the class label.
2. Each attribute set $\mathbf{x}$ is a collection of fixed number of values that can be of different data types. Conceptually, $\mathbf{x}$ represents different attributes of an entity. E.g. if we are dealing with persons, attributes can be name, age, gender, height, etc.
3. Class label denotes which group the particular attribute set belongs to.
4. The following slide shows an example of data with attribute sets and class labels.



Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber- nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

Figure: An example of dataset in classification problems. In this example, attribute set consists of seven items, ‘Body Temperature’ to ‘Hibernates’, but ‘Name’ is not an attribute.

Now we can define the classification problem.

Definition: Classification is the task of learning a target function f that maps each attribute set $\mathbf{x}$ to one of the predefined class labels y.

Two points should be noted here,

1. In the classification problem, we assume that there is an unknown functional relationship $y = f(\mathbf{x})$
2. The mechanism of estimating the function f from data is called 'learning'.
3. The estimated function, usually denoted as $\hat{f}$, is called the 'classification model', or simply a 'classifier'.

Why Classification?

Classification models can be used in two different scenarios.

1. **Descriptive Modeling**: In descriptive modeling, we are interested in identifying the relevant attribute of each class.
   - In the previous example, one could ask how mammals differ from reptiles. A classification model helps in answering this problem.
2. **Predictive Model**: Suppose we have an item whose attributes are not in the dataset, then which class does the item belong? We can use the classification model, input the attributes, and get the estimated value of the class label. This type of analysis is called prediction.

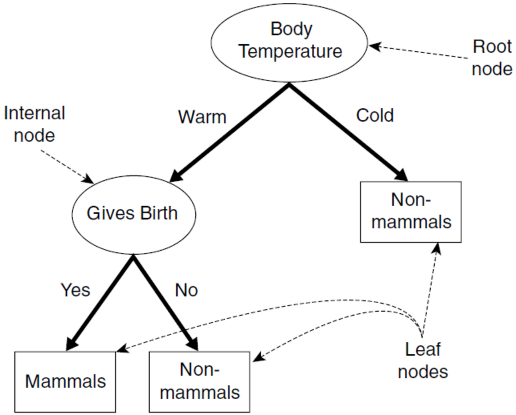


Figure: A decision tree for mammal classification

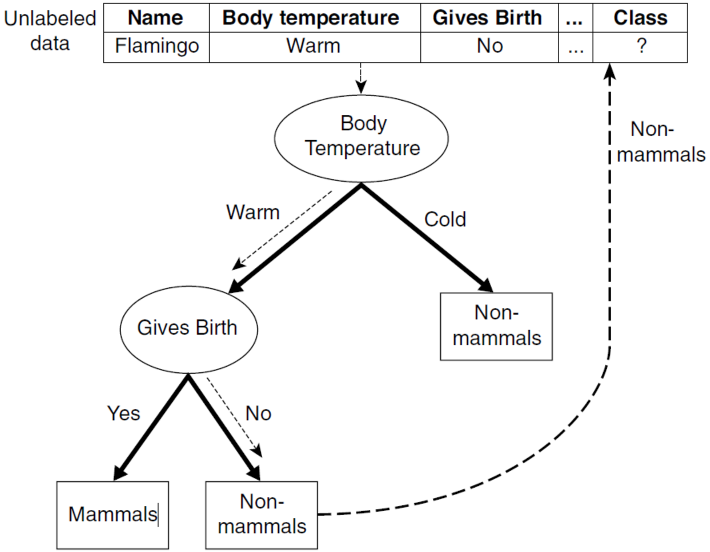


Figure: Prediction using a decision tree classifier

General Method of Building Classification Models

Given a dataset D that consists of tuples $\{(\mathbf{x}_1, y_1), \ldots, \mathbf{x}_N, y_N\}$,

1. Separate D into two parts, the training set and the test set.
2. Build the classification model $\hat{f}$ using the training set.
3. Apply $\hat{f}$ to the training set, and calculate the accuracy parameters. If accuracy is satisfactory, $\hat{f}$ is the required model. Otherwise, re-adjust the parameters of the learning algorithm and go to step 2.

Note that this method is used not only for learning classifiers from data, but in almost all machine learning problems. In addition, there are other methods (e.g. cross-validation) to split D for training and testing.

Assessing Classification Accuracy

There are many indicators of that describe the quality of a classifier. We will consider only the accuracy defined as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

The opposite of accuracy is error rate defined as

$$\text{Error rate} = \frac{\text{Number of wrong predictions}}{\text{Total number of predictions}}$$

Confusion Matrix – a 2-Class Example

Table: Confusion Matrix

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

Here, f_{ij} denotes the number of instances when an object belonging to class i is predicted as class j by the classifier. Thus,

$$\text{Accuracy} = \frac{f_{11}+f_{00}}{f_{11}+f_{10}+f_{01}+f_{00}}$$

$$\text{Error rate} = \frac{f_{10}+f_{01}}{f_{11}+f_{10}+f_{01}+f_{00}}$$

Note: Usually, f_{11}, f_{10}, f_{01} and f_{00} are also called True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN), respectively.

There are other important parameters of accuracy too:

$$\text{Sensitivity (or, true-positive rate)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{Specificity (or, true-negative rate)} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

etc.

- Accuracy and error rates are not the sole parameters to assess a classifier. Depending upon the context, we may desire to maximize or minimize one among TP, TN, FP and FN.
  - Think of those situations!
- Also, beware that the Data mining and Machine Learning communities may be using different terminology for the same concept.

It's a classwork time!!!

Decision-Tree Based Classification

How a Decision Tree Works

- The root and internal nodes of the tree correspond to the attributes
- The branches going out from a node corresponding to values of the attribute that separate the items under consideration.
- The leaves correspond to the class labels.

Hunt's Algorithm – the Preliminary Method

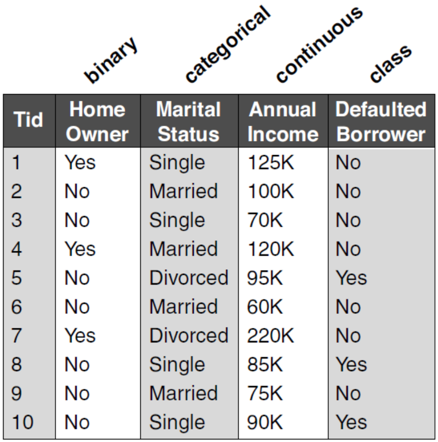
The classic algorithm for decision tree induction is Hunt's algorithm which forms the basis for modern algorithms like ID3, C4.5, and CART. Following are the main steps of the algorithm.

Hunt's Algorithm

D_t: the set of training records that correspond to node t

$y = \{y_1, y_2, \ldots, y_n\}$: the class labels

1. If all the records in D_t belong to the same class y_t, then t is a leaf node labeled as y_t.
2. If D_t contains records that belong to more than one class,
    - 2.1 an *attribute test condition* is selected to partition the record into smaller subsets.
    - 2.2 a child node is created for each outcome of the test condition
    - 2.3 records on D_t are distributed to the children based on the outcomes.
    - 2.4 then this algorithm (i.e. steps 1 and 2) are recursively applied to each child node.


binary
categorical
continuous
class

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Figure: Data for illustrating the Hunt's algorithm

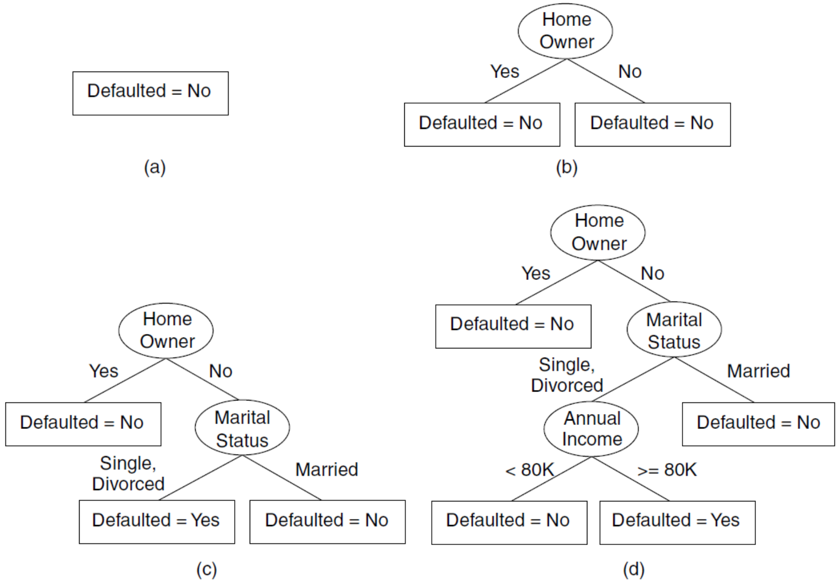


Figure: Illustrating the Hunt's algorithm

Limitations of Hunt's Algorithm

- Hunt's algorithm is too simplistic to handle real data
    - It assumes that each combination of the attributes have unique class labels
- There would could be two some situations not handled by Hunt's algorithm
    1. If the records corresponding to a node are similar attributes, it may not be possible to split the node further. In this case, the node is declared a leaf with the same class label as the majority class.
    2. It is also possible that some of the child nodes created in step 2 of the algorithm are empty. This is because the node under examination may not all attribute values being tested. In such cases, the node under examination is declared as a leaf with same class label as the majority class.
- Incorporating above issues in Hunt's algorithm, together with the method of splitting nodes gives the Decision Tree Induction algorithm.

It's a classwork time!!!

Methods for Expressing Attribute Test Conditions

- At each node, the decision tree induction algorithm selects an attribute to split the node further.
- As the attributes can be of different types, depending upon the attribute selected, there will be different method for splitting the nodes.
- This is explained in the following slides.

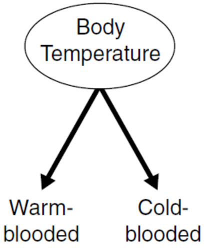


Figure: Branching based on a binary attribute

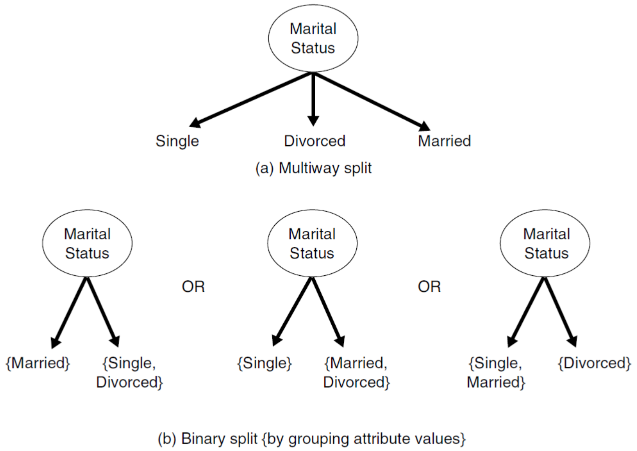


Figure: Branching based on a nominal attribute

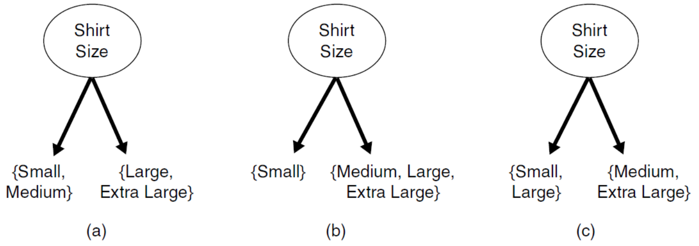


Figure: Branching based on an ordinal attribute

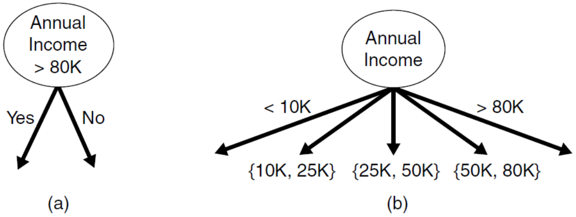


Figure: Branching based on a continuous attribute

Measures for Selecting the Best Split

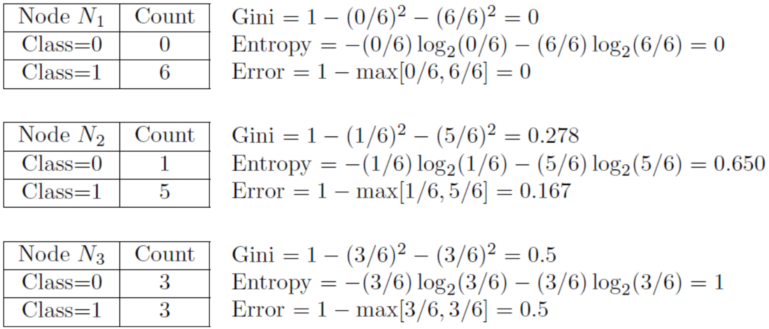
Suppose there are c class labels $\{1, 2, \ldots, c\}$ and p_i is the probability that a randomly selected element belongs to class i. Following are mostly used measures used for splitting a node in a decision tree.

$\text{Entropy} = -\sum_{i=1}^{c} p_i \log_2 p_i$
(while calculating entropy, we take $0 \log_2 0 = 0$)

$\text{Gini} = 1 - \sum_{i=1}^{c} p_i^2$

$\text{Classification error} = 1 - \max_i\{p_i\}$

Examples



Node N_1	Count
Class=0	0
Class=1	6

$$\text{Gini} = 1 - (0/6)^2 - (6/6)^2 = 0$$
$$\text{Entropy} = -(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$$
$$\text{Error} = 1 - \max[0/6, 6/6] = 0$$

Node N_2	Count
Class=0	1
Class=1	5

$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$
$$\text{Entropy} = -(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$$
$$\text{Error} = 1 - \max[1/6, 5/6] = 0.167$$

Node N_3	Count
Class=0	3
Class=1	3

$$\text{Gini} = 1 - (3/6)^2 - (3/6)^2 = 0.5$$
$$\text{Entropy} = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$$
$$\text{Error} = 1 - \max[3/6, 3/6] = 0.5$$

Figure: Example: calculating impurity measures for binary classification

Examples

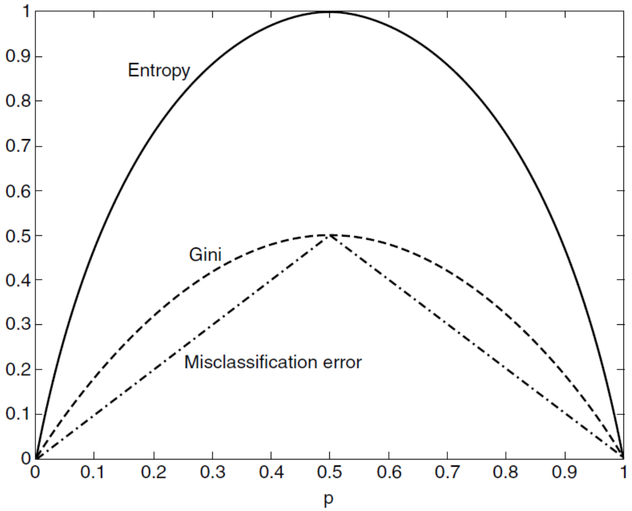


Figure: The three impurity measures give similar rankings according to purity – example for the binary classification case

Splitting a Node Based on Purity

- Hunt's algorithm (and its extensions) try split nodes taking each of the attribute as the test variable.
- Among all possibilities, the attribute that splits the node with lowest impurity is selected.
- Recall that lowest impurity corresponds to the lowest values of gini, entropy and classification error.
- However, when a node is split, there are more than one child. So, we need to compute the weighted average of the measure for the child nodes.
- We using the *gain* criterion for selecting the best possible split among the candidate splits.

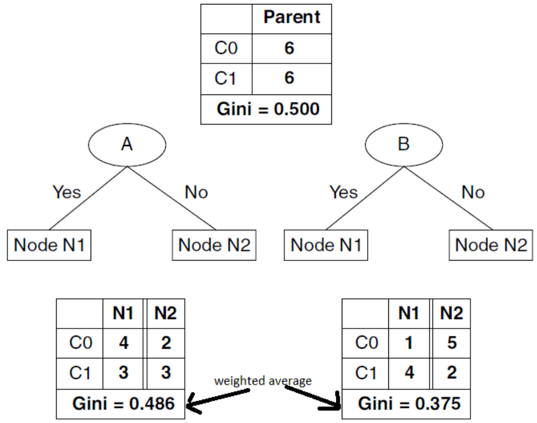
The Gain Criterion

Suppose that a parent node is split into k child nodes labeled as $v_1, v_2, \ldots, v_k$. For any impurity measure $I(.)$, the *gain* is defined as

$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

where $N(v_j)$ and N are the number of records in node v_j and the parent node, respectively.

- When we use entropy as the impurity measure, Δ is called the *information gain*.
- Information gain is used as the splitting criterion in popular decision tree algorithms like C4.5

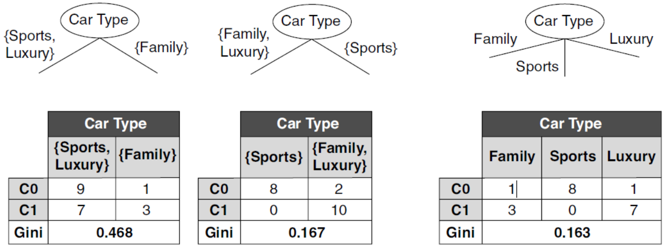


	Parent
C0	6
C1	6
Gini = 0.500	

	N1	N2
C0	4	2
C1	3	3
Gini = 0.486		

	N1	N2
C0	1	5
C1	4	2
Gini = 0.375		

Figure: Splitting binary attributes. These example shows there are two candidate attributes, *A* and *B*. In this example, *B* is preferable.



	Car Type	
	{Sports, Luxury}	{Family}
C0	9	1
C1	7	3
Gini	0.468	

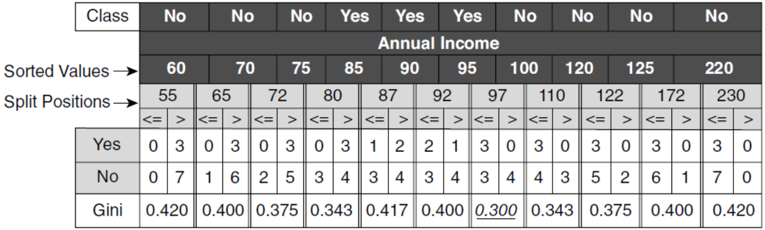
	Car Type	
	{Sports}	{Family, Luxury}
C0	8	2
C1	0	10
Gini	0.167	

(a) Binary split

	Car Type		
	Family	Sports	Luxury
C0	1	8	1
C1	3	0	7
Gini	0.163		

(b) Multiway split

Figure: Splitting nominal attributes. Figure (a) illustrate the situation when the algorithm under consideration is using binary splitting. Then, clearly the split shown at the right is better as it has low gini. As shown in figure (b), if the algorithm is using a multiway split, the attribute ‘Car Type’ with that for other attributes on the basis of resulting node purity.



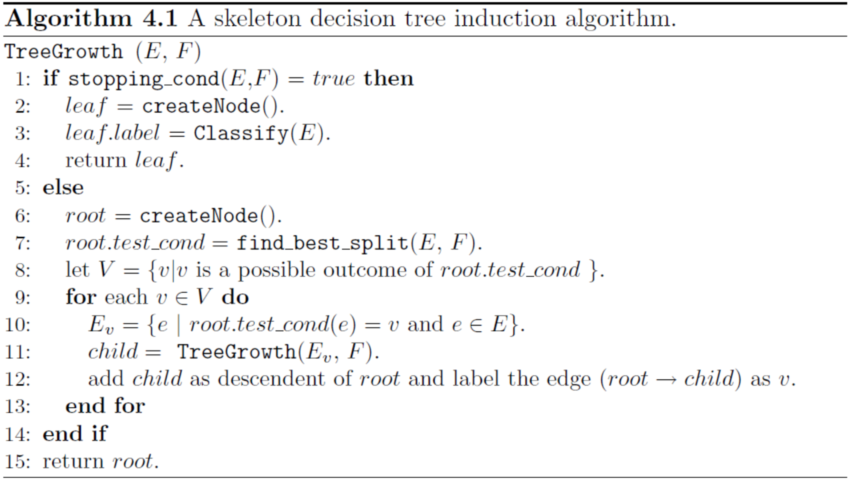
Class	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220

Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400		0.420	

Figure: Splitting continuous attributes. Each possible value of the attribute is tried as splitting point, and the point that results in lowest gini is considered. In this example, the point is '97', i.e. the branching will be based on the question 'Is Annual Income $\leq$ 97.

It's a classwork time!!!

Algorithm for Decision Tree Induction



Algorithm 4.1 A skeleton decision tree induction algorithm.

```
TreeGrowth $(E, F)$
 1: if stopping_cond$(E,F)$ $=$ $true$ then
 2:     $leaf$ = createNode().
 3:     $leaf.label$ = Classify$(E)$.
 4:     return $leaf$.
 5: else
 6:     $root$ = createNode().
 7:     $root.test\_cond$ = find_best_split$(E, F)$.
 8:     let $V = \{v | v$ is a possible outcome of $root.test\_cond$ $\}$.
 9:     for each $v \in V$ do
10:         $E_v = \{e \mid root.test\_cond(e) = v$ and $e \in E\}$.
11:         $child$ = TreeGrowth$(E_v, F)$.
12:         add $child$ as descendent of $root$ and label the edge $(root \rightarrow child)$ as $v$.
13:     end for
14: end if
15: return $root$.
```

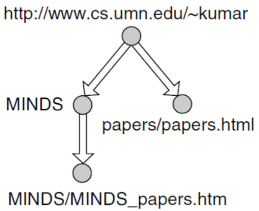
Figure: Overall idea of decisions tree induction. The inputs E and F denote the training and the attribute set.

Decision tree induction is an recursive algorithm is. Following are the major steps.

- *stopping_cond*: The recursion will stop at a particular node if that node has tolerable purity value.
- *Classify*: Once a node has acceptable purity, it will be classified as the class that is most frequent in the node, and the node will be considered a leaf. Then the recursion will backtrack to the earlier node.
- *CreateNode*: This function denotes the creation of a new node in the tree-building process. This function makes proper adjustments to the data-structure being used for implementing the tree.
- *FindBestSplit*: This function examines all candidate attributes, and also all possible decision points for each attribute. It results the combination best attribute and its decision point based on the purity measure being used.

Session	IP Address	Timestamp	Request Method	Requested Web Page	Protocol	Status	Number of Bytes	Referrer	User Agent
1	160.11.11.11	08/Aug/2004 10:15:21	GET	http://www.cs.umn.edu/ ~kumar	HTTP/1.1	200	6424		Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:34	GET	http://www.cs.umn.edu/ ~kumar/MINDS	HTTP/1.1	200	41378	http://www.cs.umn.edu/ ~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:15:41	GET	http://www.cs.umn.edu/ ~kumar/MINDS/MINDS _papers.htm	HTTP/1.1	200	1018516	http://www.cs.umn.edu/ ~kumar/MINDS	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
1	160.11.11.11	08/Aug/2004 10:16:11	GET	http://www.cs.umn.edu/ ~kumar/papers/papers. html	HTTP/1.1	200	7463	http://www.cs.umn.edu/ ~kumar	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
2	35.9.2.2	08/Aug/2004 10:16:15	GET	http://www.cs.umn.edu/ ~steinbac	HTTP/1.0	200	3149		Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7) Gecko/20040616

(a) Example of a Web server log.

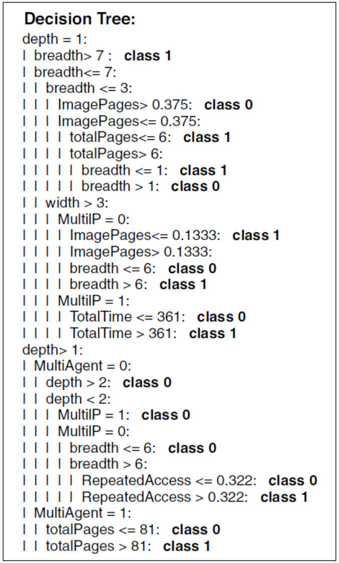


(b) Graph of a Web session.

Attribute Name	Description
totalPages	Total number of pages retrieved in a Web session
ImagePages	Total number of image pages retrieved in a Web session
TotalTime	Total amount of time spent by Web site visitor
RepeatedAccess	The same page requested more than once in a Web session
ErrorRequest	Errors in requesting for Web pages
GET	Percentage of requests made using GET method
POST	Percentage of requests made using POST method
HEAD	Percentage of requests made using HEAD method
Breadth	Breadth of Web traversal
Depth	Depth of Web traversal
MultiIP	Session with multiple IP addresses
MultiAgent	Session with multiple user agents

(c) Derived attributes for Web robot detection.

Figure: Example – Input Data for web robot detection



Decision Tree:

```
depth = 1:
| breadth> 7 :   class 1
| breadth<= 7:
| | breadth <= 3:
| | | ImagePages> 0.375:   class 0
| | | ImagePages<= 0.375:
| | | | totalPages<= 6:   class 1
| | | | totalPages> 6:
| | | | | breadth <= 1:   class 1
| | | | | breadth > 1:   class 0
| | width > 3:
| | | MultiIP = 0:
| | | | ImagePages<= 0.1333:   class 1
| | | | ImagePages> 0.1333:
| | | | breadth <= 6:   class 0
| | | | breadth > 6:   class 1
| | | MultiIP = 1:
| | | | TotalTime <= 361:   class 0
| | | | TotalTime > 361:   class 1
depth> 1:
| MultiAgent = 0:
| | depth > 2:   class 0
| | depth < 2:
| | | MultiIP = 1:   class 0
| | | MultiIP = 0:
| | | | breadth <= 6:   class 0
| | | | breadth > 6:
| | | | | RepeatedAccess <= 0.322:   class 0
| | | | | RepeatedAccess > 0.322:   class 1
| MultiAgent = 1:
| | totalPages <= 81:   class 0
| | totalPages > 81:   class 1
```

Figure: Decision tree for web robot detection (shown as rules)

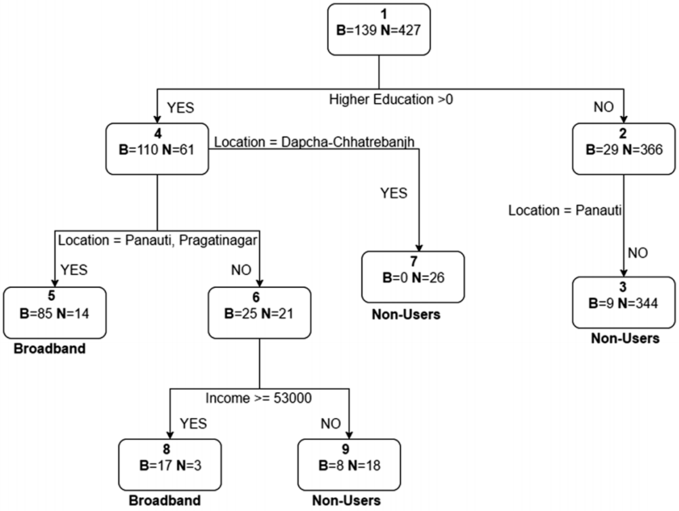


Figure: Decision tree describing household broadband use (Pandey and Raj, 2016). 'B' denotes broadband-using households and 'N' non-using.

Overfitting – a General Discussion

What is Overfitting?

- A classification (or a regression) model is said to be *overfitted* if
  1. The model has high accuracy in the test data set, but
  2. the model has low accuracy in data not included in the training set.
- Overfitting would not be a problem if
  1. If the data set is completely noise free, and
  2. If the data consists enough items corresponding to all the categories
- However, in reality, data cannot be noise free. In addition, data may not represent all possible variation in the actual population. Thus, inferences made using an overfitted model would not be reliable.

Overfitting Example

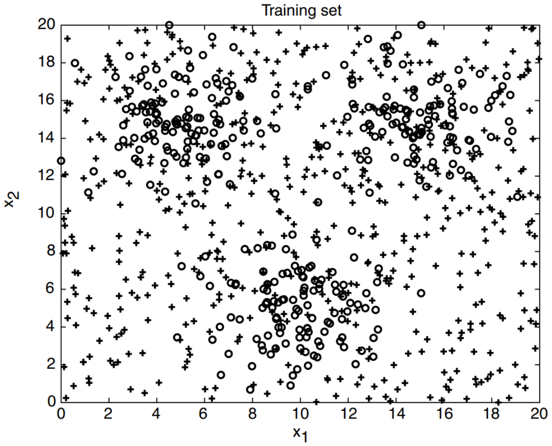


Figure: A dataset of 3000 points with 1200 points in the 'o' and 1800 points in the '+' class

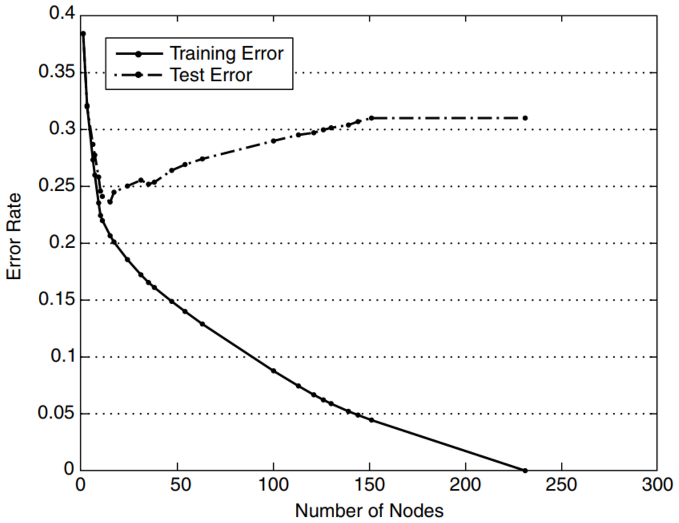


Figure: Relationship of training and test error with the number of nodes in the decision tree (70% of the data points were included in the training and 30% in the test set.)

Avoiding Model Overfitting – Incorporate Model Complexity

- Algorithms for building ML models try to minimize the training error
- While minimizing the training error, it is likely that model becomes more complicated.
  - e.g. Assume that in a classification problem, each class is characterized by a unique combination of attributes. Then it is possible to build an error free classifier in the training set by a decision tree that has nodes corresponding to every attribute and branches corresponding to every attribute-value.
- The basic idea is thus to somehow incorporate model's complexity while calculating the accuracy of the model
  - e.g. For decision trees, a simple measure of model-complexity is the number of leaf nodes.

Example: Incorporating Model Complexity

For decision trees, we could define the *pessimistic error* as

$$e_g(T) = \frac{e(T) + \Omega T}{N_t}$$

where $e(T)$ is the classification error of the tree T in the training set, $\Omega(T)$ is the complexity of the tree (which could be defined as cl, where c is a constant and l is the number of leaves), and N_t is the number of records in the training data.

- The basic idea in this example is to generate different decision trees and select the tree T that has minimum $e_g(T)$.

A Digression ...

- The procedure in the previous slide is an example of a broader technique called *regularization*.
- Regularization is used mainly to tackle *ill-posed problems*, and also for avoiding overfitting.
- The general idea of regularization is to minimize the regularized error function defined as

  $$\mathscr{E}_R(\hat{f}) = \mathscr{E}(\hat{f}) + \lambda R(\hat{f})$$

  where $\mathscr{E}(\hat{f})$ is the error of the model $\hat{f}$, λ is the *regularization parameter*, $R(\hat{f})$ is the *regularization function*, which in our case, is any measure of model complexity. The value of the regularization parameter λ is determined either analytically, or through empirical techniques like cross-validation (which will be discussed soon.)

An Important Practical Terminology – Hyperparameter Tuning

- In practice, the structure of the model, including its complexity, is defined by a set of parameters intrinsic to the classifier/regression model. These parameters are called *hyperparameters*
- For example, decision trees have hyperparameters like
  1. Maximum allowable tree depth
  2. The branching factor
  3. The minimum number of records corresponding to a node required to split the node.
- Another example, neural networks have hyperparameters like the number of layers, the number of nodes in each layer, etc.

The Practical Challenge in Avoiding Overfitting

- The basic idea of avoiding overfitting is to find the values of the parameters that minimize the test error (or the cross-validation error, if we are using cross-validation)
- In ML models, the parameters that define a model's complexity are usually large in number. In addition, each parameter could have a large range of values. This results in extremely large combinations of hyperparamters.
  - **Classwork**: Think of examples please...
- The complications increases because of the fact that even for a given values of the hyperparameters, ML algorithms use significantly long time to build the desired model.
- **Conclusion**: The availability of time and computing resources should guide how fine we need to do the hyperparameter tuning.

Cross-Validation for Hyperparmeter Tuning

The overall steps in a k-fold cross-validation are following

1. Divide the data set into k equal sized subsets
2. For each combination of the hyperparameter values, do
    - 2.1 Select one subset as the test set, and remaining $k - 1$ subsets jointly as the training set.
    - 2.2 Fit the model in the training set, calculate and save test error.
    - 2.3 Repeat above two steps for all possible combination of training and test sets.
    - 2.4 Calculate average test error for all the k combinations above.
3. Select the values of hyperparameters that give the minimum average test error in above step.
4. Fit the model in the whole dataset using the hyperparameter values selected in above step.

Bayesian Classification

Bayes' Theorem

- Let H deonote a hypothesis and $\mathbf{X}$ the evidence variable.
- $P(H)$ is the probability that the hypothesis H is true , i.e.the *prior probabilty* of H.
- $P(\mathbf{X})$ is the probability that $\mathbf{X}$ occurs, i.e.*prior probability* of H.
- $P(H \mid \mathbf{X})$ is the probability that H is true given X has occurred, i.e. the posterior probability of H conditioned on $\mathbf{X}$
- $P(\mathbf{X} \mid H)$ is the probability that X occurs given H is true, i.e. the posterior probability of X conditioned on H.

Then,

$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

Naive Bayes Classifier

- Let there be D tuples in the set, with each tuple having n attributes, i.e. $\mathbf{X} = (x_1, x_2, \ldots, x_n)$.
- Let there are m classes $C_1, C_2, \ldots, C_m$. Then our problem has m hypothesis of the form

  H_i:The object being considered belongs to class C_i
- We are interested in the hypotheses

  "Given the object has attribute $\mathbf{X}$, what is the probability that it belongs to a particular class C_i?"

  In other words, we are interested in the conditional probabilities $P(C_i \mid \mathbf{X})$

- The Naive Bayes classifier predicts the tuple X belongs to a class C_i if and only if

$$P(C_i \mid \mathbf{X}) > P(C_j|\mathbf{X}) \text{ for all } j \neq i.$$

- Note that in the Bayes' theorem, we need the prior probabilities $P(\mathbf{X})$ and $P(H)$ (i.e. $P(C_i)$ in the classification problem) which are generally unknown.
  - We do not require to calculate the prior probability $P(\mathbf{X})$ because it is constant for across the classes. So, we only require to calculate the numerator terms $P(\mathbf{X} \mid H)P(H)$.
  - Regarding $P(H)$, the naive Bayes' algorithm estimates it from the given sample.

- If the dataset has a large number of attributes, computation of the conditional probabilities $P(\mathbf{X} \mid C_i)$ becomes computationally expensive. The naive Bayes' method makes the **assumption of conditional independence** which is as follows.

$$\begin{aligned} P(\mathbf{X} \mid C_i) &= \prod_{k=1}^{n} P(x_k \mid C_i) \\ &= P(x_1 \mid C_i) \times P(x_2 \mid C_i) \times \ldots \times P(x_n \mid C_i) \end{aligned}$$

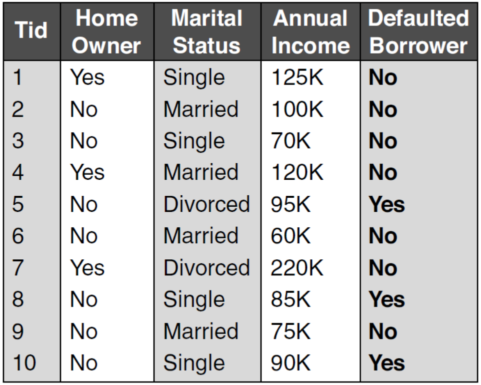
In simple words, this **naive Bayes assumption** states that the attributes are independent with each other for the same class.

- The question now remains is how to calculate the conditional probabilities $P(x_k \mid C_k)$
  - If the attributes are discrete, then $P(x_k \mid C_k)$ equals the number of tuples of class C_i having the k'th attribute equal to x_k divided by the number of tuples of class C_i.
  - If the attributes are real numbers, we assume that they follow certain probability distributions, usually the Gaussian distribution. Under the Gaussian assumption,
    $$P(x_k \mid C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$
    where
    $$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Finally, **the naive Bayes algorithm classifies according to the following rule**.

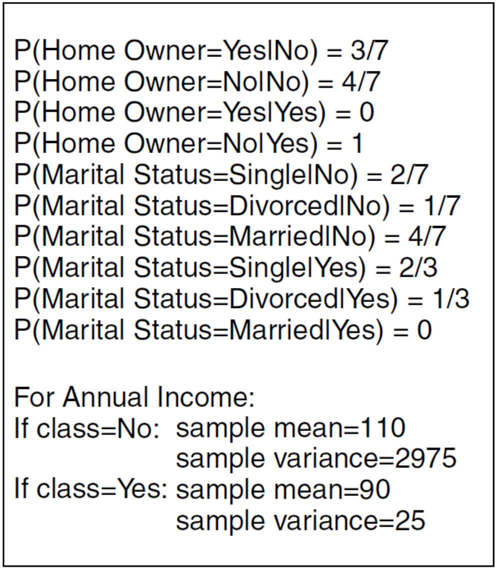
> "The tuple $\mathbf{X}$ is predicted to class C_i if and only if $P(\mathbf{X} \mid C_i)P(C_i) > P(\mathbf{X} \mid C_j)P(C_j)$ for all $j \neq i$"

Naive Bayes Example from Tan et al. 2006



Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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**

The first step is to compute and save the conditional probabilities. Note that this example uses the Gaussian distribution assumption for annual income.



P(Home Owner=Yes|No) = 3/7
P(Home Owner=No|No) = 4/7
P(Home Owner=Yes|Yes) = 0
P(Home Owner=No|Yes) = 1
P(Marital Status=Single|No) = 2/7
P(Marital Status=Divorced|No) = 1/7
P(Marital Status=Married|No) = 4/7
P(Marital Status=Single|Yes) = 2/3
P(Marital Status=Divorced|Yes) = 1/3
P(Marital Status=Married|Yes) = 0

For Annual Income:
If class=No: sample mean=110
sample variance=2975
If class=Yes: sample mean=90
sample variance=25

Now we can use the naive Bayes classifier. Suppose a person does not own a home, is not married, and has annual income of \$120k, i.e. $\mathbf{X} = \{\text{No,No,\$120K}\}$. Is she a defaulted borrower?

$$\begin{aligned} P(\mathbf{X}|\texttt{No}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{No}) \times P(\texttt{Status} = \texttt{Married}|\texttt{No}) \\ &\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{No}) \\ &= 4/7 \times 4/7 \times 0.0072 = 0.0024. \end{aligned}$$

$$\begin{aligned} P(\mathbf{X}|\texttt{Yes}) &= P(\texttt{Home Owner} = \texttt{No}|\texttt{Yes}) \times P(\texttt{Status} = \texttt{Married}|\texttt{Yes}) \\ &\quad \times P(\texttt{Annual Income} = \$120\text{K}|\texttt{Yes}) \\ &= 1 \times 0 \times 1.2 \times 10^{-9} = 0. \end{aligned}$$

In addition, $P(Yes) = 0.3$ and $P(No) = 0.7$ for the class label. Since $P(\mathbf{X} \mid \text{No})P(\text{No}) > P(\mathbf{X} \mid \text{Yes})P(\text{Yes})$, the person is not a defaulted borrower.

Laplace Smoothing

- Since the naive Bayes' classifier depends on a product of probabilities, occurrence of zero probability values could make classification difficulty. Because of the zero values, it may happen that for some $\mathbf{X}$, the probability $P(\mathbf{X} \mid C_i) = 0$ for all classes C_i.
- Thus to avoid zero probabilities, we use Laplace smoothing. The idea is to re-calculate the probability values as below.

$$P_{\text{Laplace}}(x_i) = \frac{\text{count}(x_i) + \alpha}{D + \alpha N}$$

  where α is a small number called the smoothing factor, D the number of tuples in the dataset, and N the number of attributes.
- If α is chosen very small compared to D, then we can still have zero probabilities because of limited floating-point precision of hardware. Thus, the appropriate value of α depends on D.

Classification by Back-Propagation (Neural Network)

Classification using Neural Network

- A neural network has a input layer, one or more hidden layers, and an output layer.
- The output of a node obtained as follows.
  - First, calculate the weighted sum of the values of the nodes in the previous layer plus the bias.
  - Calculate the output of the node using the logistic function. If v is the weighted sum of the node-values of the previous layer plus the bias, then the value of an output node will be

$$O = \frac{1}{1 + e^{-v}}$$

- Note that the output of a node is a real number, whereas in classification problems, the outputs need to be discrete. To use a neural network for classification, we can assign the final output(s) to either 0 or 1, whichever is near.

Gradient Descent

- Then idea of training a neural network is gradient descent: repeatedly change the weights by small amount along the direction where the rate of increase of error is minimum. i.e.

$$w_j \leftarrow w_j - \lambda \frac{\partial E(\mathbf{w})}{\partial w_j}$$

  Here, λ is a constant called the learning rate.
- The gradient descent rule is directly applicable for weights in the input layer because it is difficult to say how E changes with the change in an input-layer weight. This problem is tackled by the back-propagation algorithm.

Backpropagation - Weight Updating Rules

- For a node j at the output layer, the error is computed as
$$Err_j = O_j(1 - O_j)(T_j - O_j)$$
where, O_j is the output and T_j is the true value at node j. Note that the term $O_j(1 - O_j)$ is the derivative of the logistic function.
- For a node j in a hidden layer, the error is
$$Err_j = O_j(1 - O_j)\textstyle\sum_k Err_k w_{jk}$$
where w_{jk} is the weight of connection from node j to a node k in the higher layer.
- Weights are updated by the following rules.
$$\Delta w_{ij} = \lambda Err_j O_i$$
$$w_{ij} = w_{ij} + \Delta w_{ij}$$
- Biases are updated as following.
$$\Delta\theta_j = \lambda Err_j$$
$$\theta_j = \theta_j + \Delta\theta_j$$

The Backpropagation Algorithm

Repeat following steps until a stopping condition is met

1. Assign each weight w_{ij} to a small random number.
2. For each $(\mathbf{X}, y)$ in the input data.
   - 2.1 Calculate the output of all nodes using feed-forward on the input $\mathbf{X}$.
   - 2.2 For each layers, from the output to the input
     - 2.2.1 Update weights and biases using the gradient descent based rules.