# Lecture 3 – Classification 1

*Data Mining, Spring 2016*

Anders Hartzen, andershh@itu.dk

IT UNIVERSITY OF COPENHAGEN

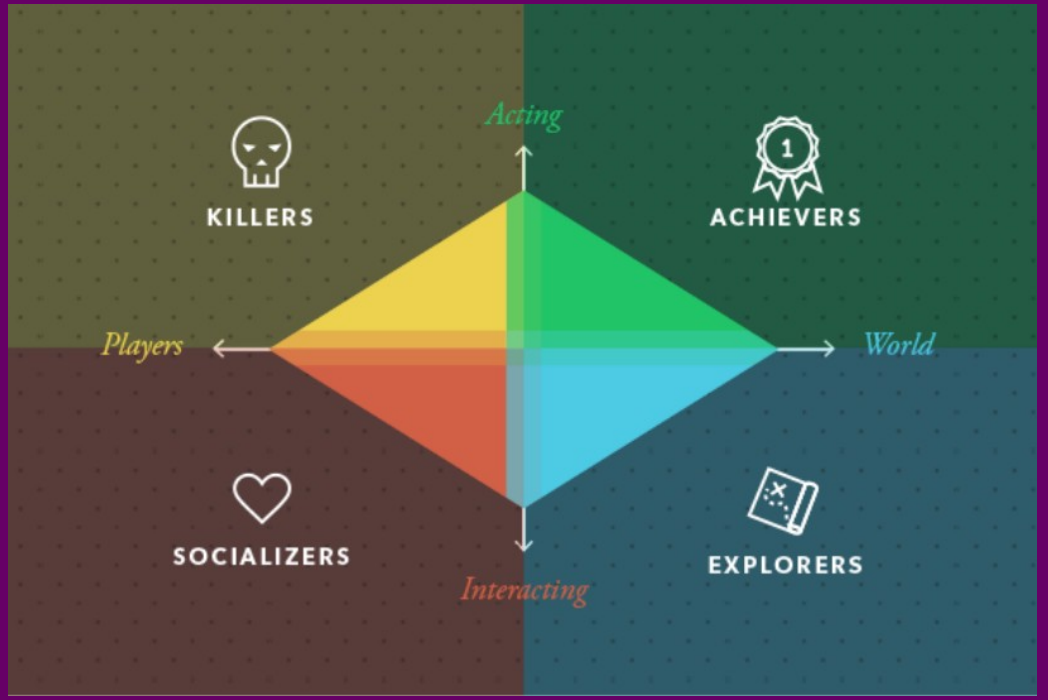
Some slides adapted from Hector Martinez of University of Malta (former ITU).

# Overview of today's lecture 1/2

- Basic Classification Concepts
- Classification Algorithms
  - Decision Tree Induction
    - Algorithm
    - Attribute Selection Measures
    - Tree Pruning
  - K-Nearest Neighbor Algorithm
- Evaluating Classification Models
  - Basic Metrics
  - Comparing Classification Models

*Basic Classification Concepts*

# Example: Bartle's taxonomy of players



# Prediction Problem

- Imagine that you are the head of shop
- As part of managing your shop you keep a database with basic information about your customers (e.g. name, age, work position etc.) and how they spend money in your shop
  - After doing customer research you have been able to divide your customers into different categories (e.g. customer type)
- Two problem scenarios
  - Based on your existing data, is there any way you can predict whether a new customer will be one type or another?
  - You want to hold a sale in the near-future, is there any way you could predict how much each different customer type will spend?

# Prediction Problem

- These two problem scenarios are examples of prediction problems, i.e. based on existing data you want to predict something about the future (e.g. future customers or future sales)
- Today we will talk about one of the main areas of prediction problems – namely classification (1st problem scenario)

# Recap from Last week: Nominal Attribute

- Nominal
  - *"of, relating to, or constituting a name"* Merriam-Webster Dictionary
- Symbol or name of *things*
  - e.g. code or category
- No meaningful order between possible values for the nominal attribute
- Also known as **categorical** or **enumeration**

| Name | Address | Position |
|---|---|---|
| John Doe | Happy Road 2 | Student |
| Jane Doe | Spring Way 1 | Student |
| Joan Petersen | Sunset Blvd. | Professor |

# What is Classification?

- Data analysis where you based on existing data try to extract model describing a specific nominal attribute, e.g. a data class
  - Data class example: Safe/Unsafe loaner in a loan applicant database, player type, customer type
  - Specific nominal attribute we want to predict known as the *class label*
- Such models are named *classifiers* because they can be used to classify new data with no information on specific nominal attribute
  - I.e. based on other attributes try to predict the class label of the unseen data object

# What is Classification?

- Examples of generated models could be
  - Decision trees
  - Rules
  - Function mapping a tuple to a class label
  - IF-THEN-ELSE rules

# What is Classification?

- *Classification* only deals with predicting nominal attributes, i.e. categorical labels
  - E.g. Safe / Unsafe loaner; customer type; player type
- This is different from *numeric prediction*, which deals with predicting numerical values
  - Model created is called a predictor
  - Found via regression analysis (e.g. least-squared error)
- Today we will focus on classification

# The Classification process

- Overall two general steps in classification
  - Learning step
    - Is when we construct our classifier, by analyzing/"learning" from data
  - Classification step
    - Is when we use our classifier to predict for some given data
    - Crucial first step: estimating classifier accuracy
- Both steps need data
  - *Training data* needed for the learning step (also known as *test tuples*)
  - *Test data* needed for the classification step, in order to estimate the accuracy of the classifier

# The Classification process

- Both the training and test data has values for the nominal attribute we want to be able to predict
  - This is an example of *supervised learning*, because we are providing data on which class each training/test tuple belongs to
  - *Unsupervised learning*, is when we do not provide (or know) in advance information on where each training tuple belongs.
    - Example: clustering, frequent pattern mining
- Question: Why do we need to have both a training and test data set? Why not just estimate accuracy using the training set?

# The Classification process

- Both the training and test data has values for the nominal attribute we want to be able to predict
- Question: Why do we need to have both a training and test data set? Why not just estimate accuracy using the training set?
  - Problem: Overfitting the data
    - A classifier that overfits data incorporates particular fluctuations / anomalies that makes it less suitable for prediction

*Classification Algorithms*

# Classification Algorithms

Two types

- Eager learners
  - Constructs a classification model straight away when presented with training data
  - When finished it is ready to classify unseen data
- Lazy learners
  - When training it essentially just stores the data for late usage
    - Therefore also known as instance-based learners, test tuple = instance
  - Only utilizes the training data when presented with unseen data to classify
    - i.e. it does not construct a model beforehand, but waits until “the last minute” to do anything

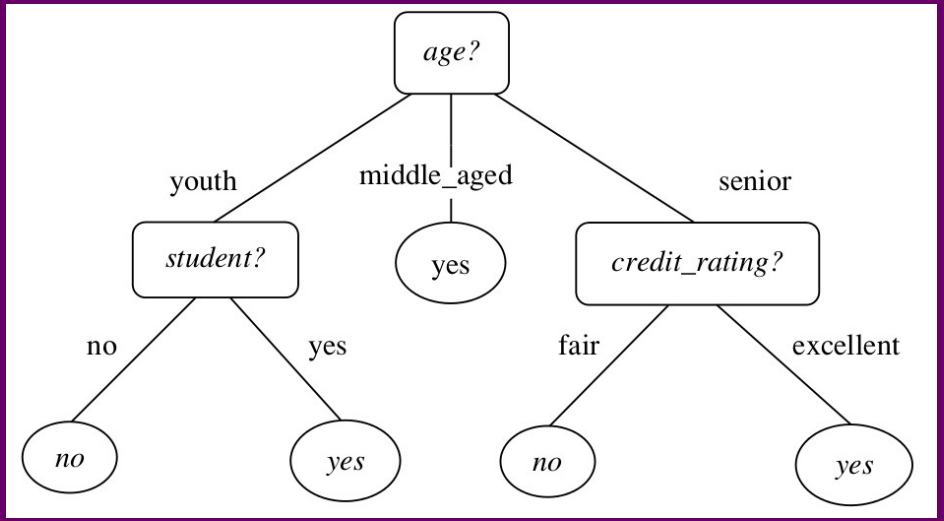
# Eager vs Lazy Learners

- Lazy learners do less work when training, but more work when classifying compared to eager learners
- Lazy learners also need to be able to store the training data in an efficient manner in memory so it can be used in the classification step
  - Especially if the data set is large...
- Today's algorithms
  - Decision Tree Induction: Eager Learner
  - K-Nearest Neighbor: Lazy Learner

*Decision Tree Induction*

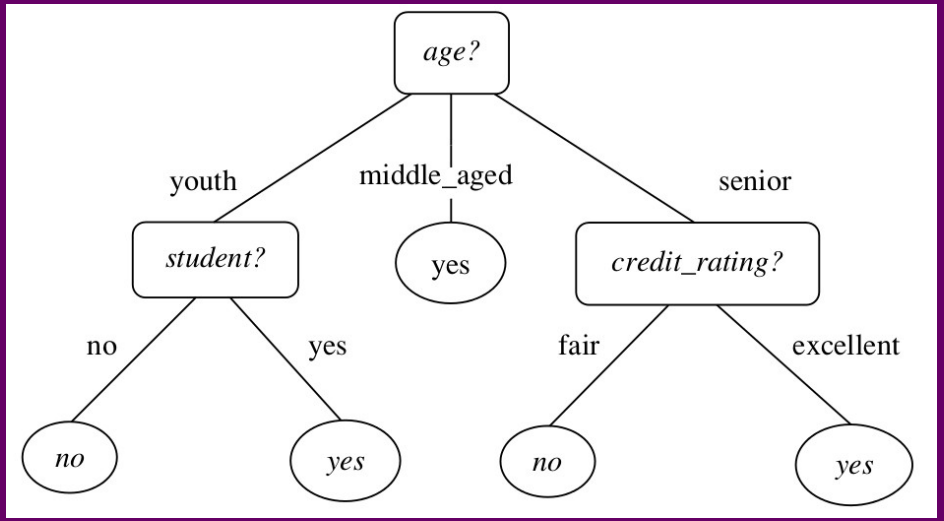
# Decision Tree

- Is a flowchart-like structure
- Each internal node (not-leaf) denotes an attribute test
  - All non-class label attributes may be tested
- Each branch represents an outcome of a test
- Each leaf holds a class label
- Top-most node known as the root node



# Decision Tree

- Decision trees can be binary
  - i.e. only two branches for each internal node
- When an unseen tuple is presented to the tree, the tuple attributes are tested in order to arrive at a class label
- Decision Tree pros
  - Human-readable
  - No domain knowledge required
  - No parameter setting required
    - Apart from attribute selection measure
  - Good for knowledge discovery
  - Typically creates high-accuracy classifiers



# ID3 Algorithm

- Eager learner which constructs a decision tree
- Algorithm overview
  - Trees created in greedy (non-backtracking) approach
    - Recursive, top-down, divide-and-conquer
  - Training data recursively divided into smaller and smaller sub-sets until classification can be made (conquer)
    - Algorithm begins at root node with full training data
  - Partitioning based on selecting “best” attribute that can create most “pure” subset from a class-label perspective
    - Subset is pure if all tuples have the same class-label

# ID3 Algorithm

- Partitioning stops when
  - All training tuples in subset has the same class label, or
  - No remaining attributes to split on (majority voting then used for leaf), or
  - No more training tuples left
- Three “ingredients” needed to run the algorithm
  - Training data partition (at first full training data set)
  - List of attributes (other than the class label)
  - Attribute selection measure (important!)
    - Used to determine what attribute to test on when dividing the test data into subsets
      - Selected attribute known as the *splitting criterion* or the *splitting attribute*
    - More on this later in this lecture

# ID3 Algorithm Step-for-step 0/8

Note: age is a nominal attribute (**not** numerical), with possible values:
<=30
31...40
>40

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

# ID3 Algorithm Step-for-step 1/8

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |


- Create a Node N, then check if all tuples in current data partitioning has same class label (false), and check whether there are no possible splitting criterion left (false)

# ID3 Algorithm Step-for-step 2/8

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

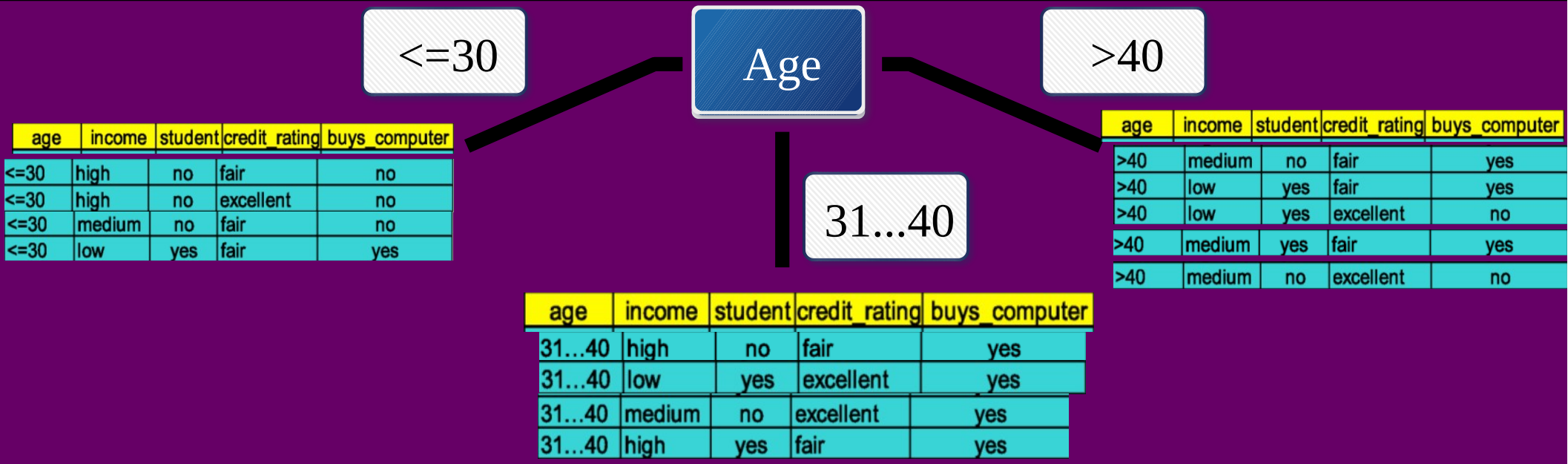

- We then apply our attribute selection measure to find the best splitting criterion

# ID3 Algorithm Step-for-step 3/8

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31…40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31…40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31…40 | medium | no | excellent | yes |
| 31…40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |


- The attribute selection measure returns age as splitting criterion.

# ID3 Algorithm Step-for-step 4/8



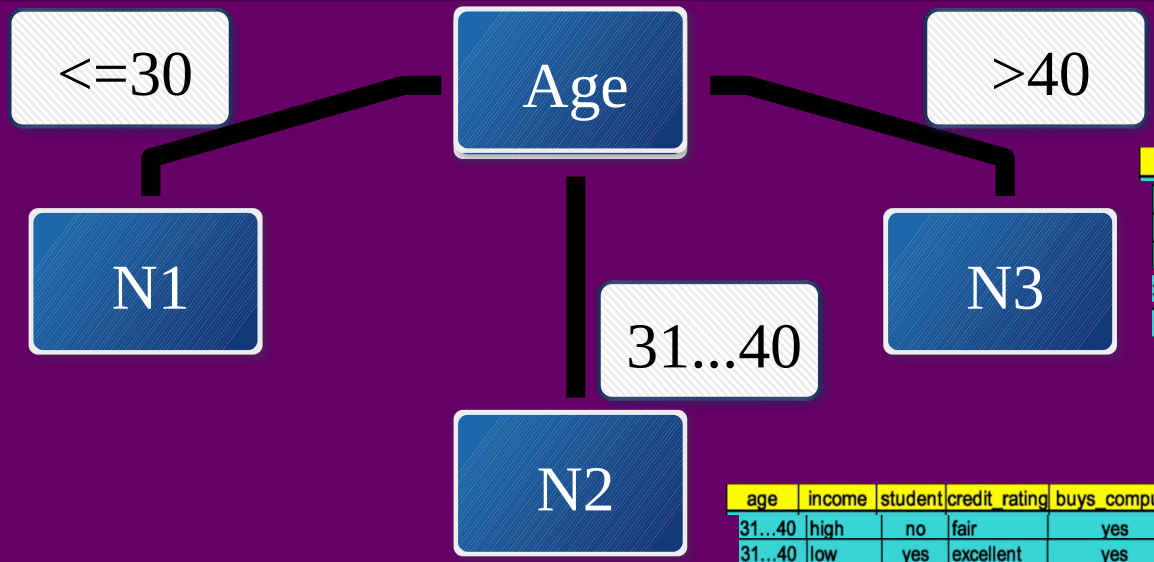

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| 31...40 | high | no | fair | yes |
| 31...40 | low | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| >40 | medium | yes | fair | yes |
| >40 | medium | no | excellent | no |

- Based on each Age value we split the data into three subsets, and remove age from list of possible splitting criteria.

# ID3 Algorithm Step-for-step 5/8



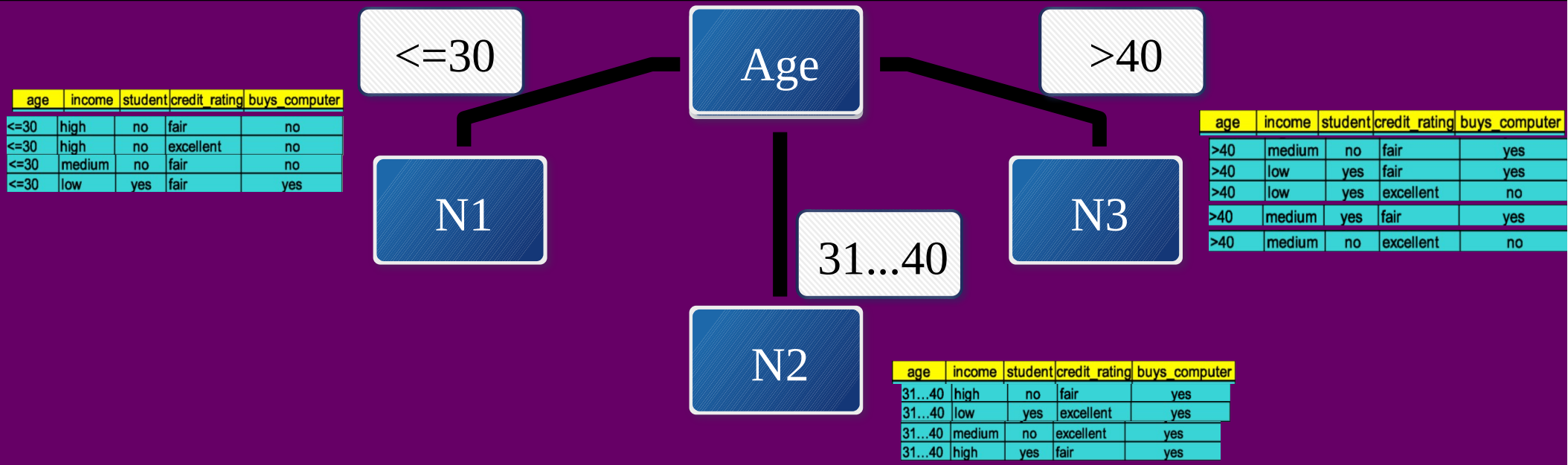

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| >40 | medium | yes | fair | yes |
| >40 | medium | no | excellent | no |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| 31...40 | high | no | fair | yes |
| 31...40 | low | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |

- We then invoke the ID3 algorithm with each subset of data and the list of available splitting criteria. This creates 3 new Nodes

# ID3 Algorithm Step-for-step 6/8



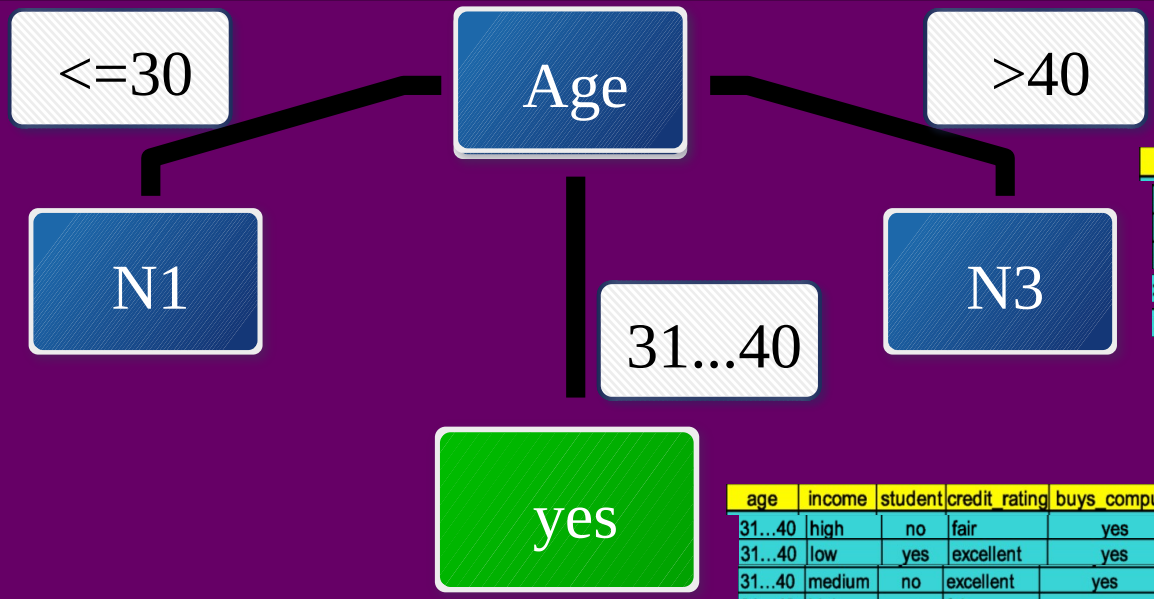

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| >40 | medium | yes | fair | yes |
| >40 | medium | no | excellent | no |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| 31...40 | high | no | fair | yes |
| 31...40 | low | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |

- The ID3 call on N2 will quickly return N2 as a leaf node, because the data subset in N2 all belong to buys_computer=yes

# ID3 Algorithm Step-for-step 7/8




| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |

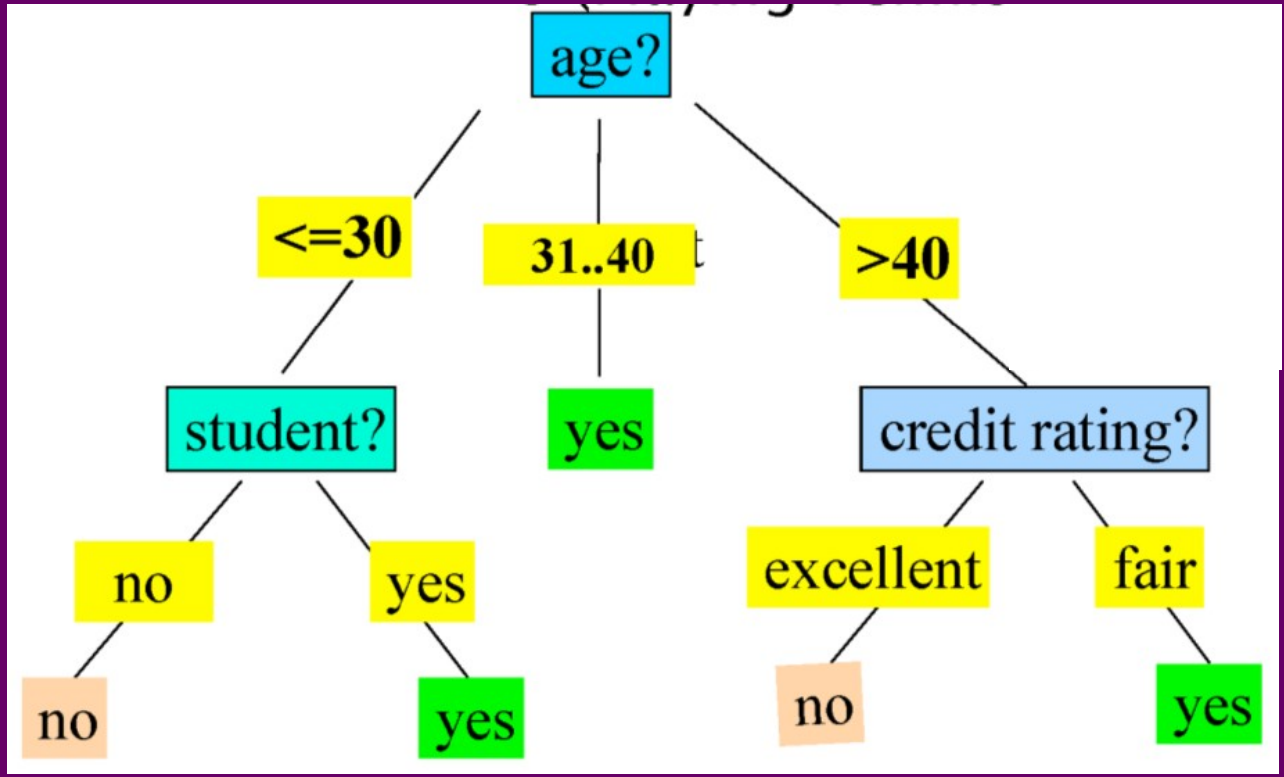
| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| >40 | medium | yes | fair | yes |
| >40 | medium | no | excellent | no |

| age | income | student | credit_rating | buys_computer |
|---|---|---|---|---|
| 31...40 | high | no | fair | yes |
| 31...40 | low | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |

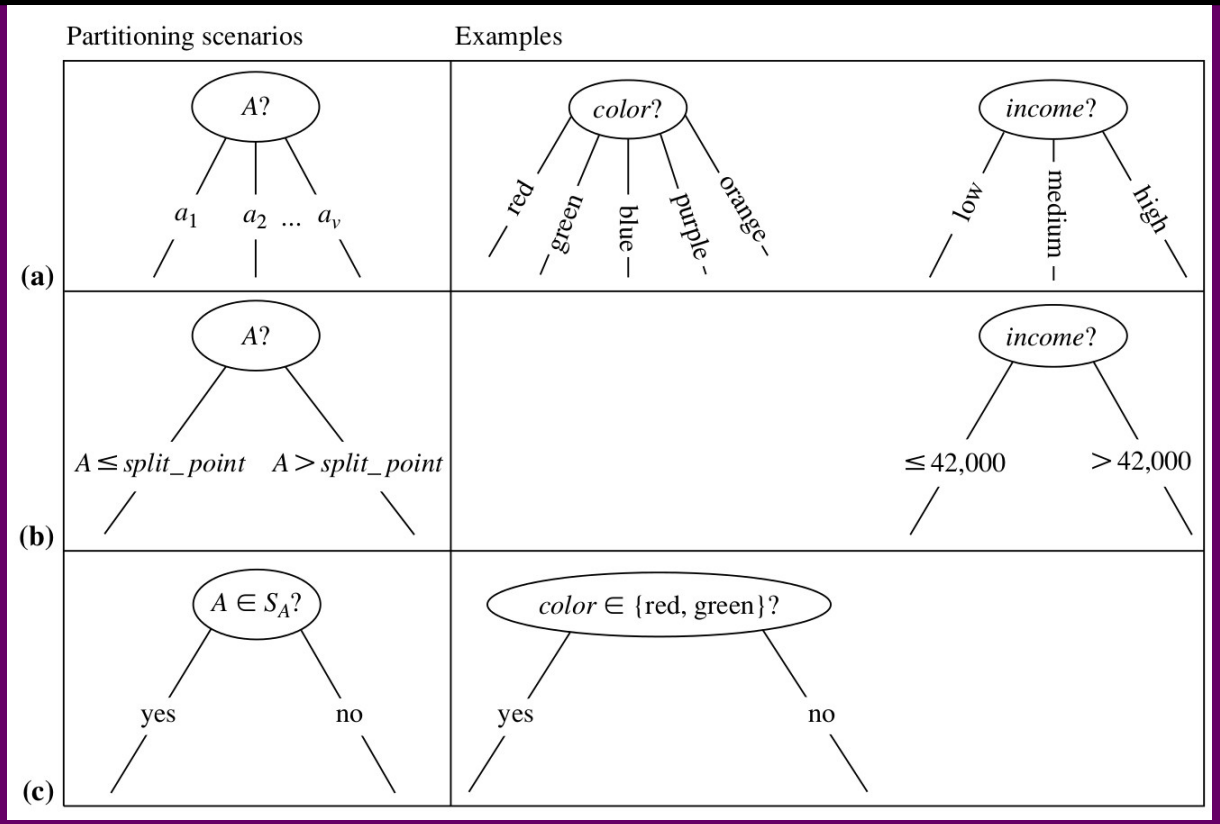
- Same is not true for N1 and N3 where further partitioning is needed, following same process as with root node.

# ID3 Algorithm Step-for-step 8/8

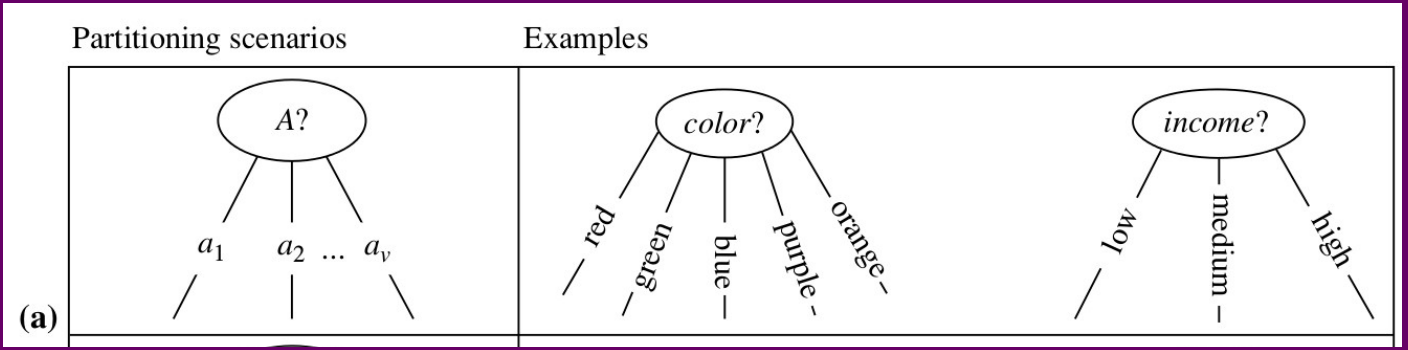
- The end result is arrived by partitioning N1's data subset using the student attribute, and partitioning N3's data subset using the credit rating attribute



# ID3 Algorithm Three Partitioning Scenarios



# ID3 Algorithm Three Partitioning Scenarios

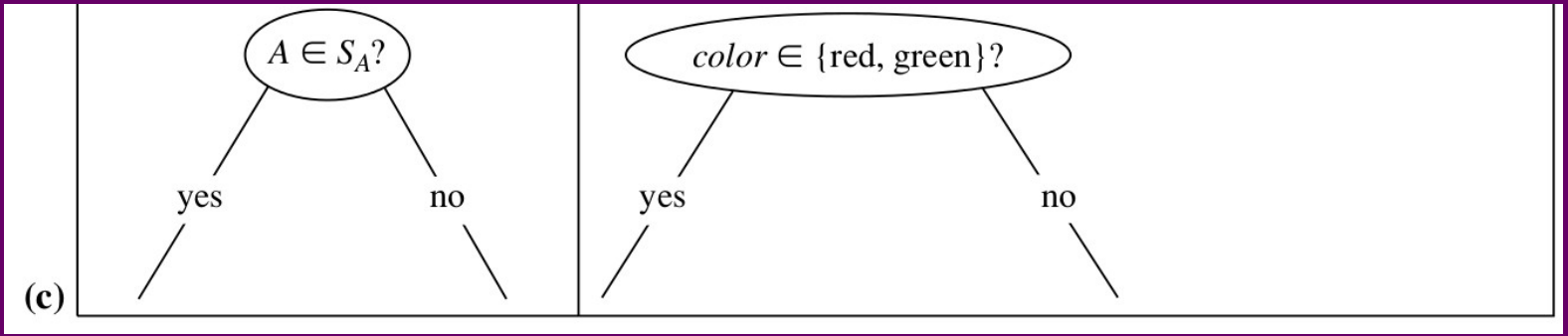



- Scenario (a) is when the splitting attribute is discrete-valued.
- Splitting attribute is removed from list of possible splitting attributes, because each subset has the same value for splitting attribute
- A branch is created for each value of the splitting attribute

# ID3 Algorithm Three Partitioning Scenarios


- Scenario (b) is when the splitting attribute is continuous-valued. The attribute selection measure returns a split point, which is used in the test at the node.
- Splitting attribute is not removed from list of possible splitting attributes
- Two branches produced
  - one denoting when the attribute value is below or equal to the split point,
  - the other when the attribute value is higher than the split point

# ID3 Algorithm Three Partitioning Scenarios




- Scenario (c) is when the splitting attribute is discrete-valued, but the decision tree should be binary. The test at the node is designed to check whether attribute value is part of splitting subset (returned by attribute selection measure)
- Splitting attribute is not removed from list of possible splitting attributes
- Two branches produced
  - one denoting when the attribute value is part of splitting subset
  - the other when the attribute value is not part of splitting subset

# Attribute Selection Measure

- Important part of Decision Tree Induction
- Our “yardstick” to determine which attribute is best to split data on
- The different Decision Tree Induction algorithms use different attribute selection measures.
- ID3 uses the Information Gain attribute selection measure

# Information Gain

- The attribute with the highest information gain is chosen as splitting criterion
  - Selected attribute minimizes the information needed to classify tuples in the resulting training data partitions
- The expected information needed to classify a tuple in partition D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

- Where m is the number of class labels, $p_i$ is the non-zero probability of a tuple belonging to class label $C_i$ and is estimated by $|C_{i,D}|/|D|$
  - $P_i$ = (number of tuples with class label $C_i$) / (number of tuples in D)

# Information Gain

- Info(D) is known as the *entropy* of partition D
- Suppose we are contemplating splitting using discrete attribute A with v distinct values
- We would want to know how much information would then be needed if we divided D into v subsets $\{D_1, D_2, D_3 \ldots D_v\}$ using A, i.e. $Info_A(D)$:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

# Information Gain

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

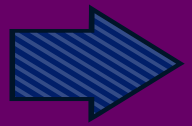
- Where $|D_j|$ is the number of tuples in partition $D_j$
- Information gain is then defined as

$$Gain(A) = Info(D) - Info_A(D).$$

# Information Gain

- Returning to our ID3 example, let us look at the information gain calculation which lead us to choose Age as the first attribute to split on
- Calculating Info(D):

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$



$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

- $P_i$ = (number of tuples with class label $C_i$) / (number of tuples in D)
  - 9 buys_computer=yes; 5 buys_computer=no; |D| = 14; two class labels hence m = 2

# Information Gain

Next step is to compute how much information would be needed if we split on Age, i.e. $Info_{Age}(D)$:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Info_{age}(D) = \frac{5}{14} \times \left( -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} \right)$$

Partition $D_1$ when age=youth,
$|D_1| = 5$; 2 buys_computer=yes;
3 buys_computer=no;
v=3 because of three values for Age

# Information Gain

Next step is to compute how much information would be needed if we split on Age, i.e. $Info_{Age}(D)$:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Info_{age}(D) = \frac{5}{14} \times \left( -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} \right)$$

$$+ \frac{4}{14} \times \left( -\frac{4}{4}\log_2\frac{4}{4} \right)$$

Partition $D_2$ when age=middle_aged,
$|D_2|$ = 4; 4 buys_computer=yes;
0 buys_computer=no;
v=3 because of three values for Age

# Information Gain

Next step is to compute how much information would be needed if we split on Age, i.e. $Info_{Age}(D)$:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right)$$

$$+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$

Partition $D_3$ when age=senior;
$|D_3|$ = 5; 3 buys_computer=yes;
2 buys_computer=no;
v=3 because of three values for Age

# Information Gain

Next step is to compute how much information would be needed if we split on Age, i.e. $Info_{Age}(D)$:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right)$$

$$+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right)$$

$$+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$

$$= 0.694 \text{ bits.}$$

Partition $D_3$ when age=senior;
$|D_2| = 5$; 3 buys_computer=yes;
2 buys_computer=no;
v=3 because of three values for Age

# Information Gain

- Now we can compute the information gain for splitting on Age to be:

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

- Similarly we can compute the information gain for the other attributes:
  - Gain(income) = 0.029 bits, Gain(student) = 0.151 bits, and Gain(credit rating) = 0.048 bits
- Age is chosen because of its largest information gain

# Information Gain for Continuous Attribute

- A is a continuous attribute
- We need to find the *split point* for A i.e. what value to split the data in two
  - First, sort the value of A in increasing order
  - Typically each midpoint between each pair of adjacent values for A is considered as possible split point
    - Midpoint = $(a_i + a_{i+1}) / 2$ is the midpoint between the values of $a_i$ and $a_{i+1}$
  - For each midpoint we compute $Info_A(D)$ with v = 2
    - D1 = subset with tuples satisfying A =< split point and D2 = subset with tuples satisfying A > split point
- Midpoint with lowest $Info_A(D)$ is chosen

# Information Gain vs Gain Ratio

- Problem: Information Gain is biased towards attributes with a large number of values (e.g. id_attribute) as choosing them results in small but pure subsets
- The C4.5 algorithm (ID3 successor) uses gain ratio to overcome this problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$

- GainRatio(A) = Gain(A) / SplitInfo(A)
- Attribute with highest GainRatio is chosen

# ID3 Pseudo code

**Algorithm: Generate_decision_tree.** Generate a decision tree from the training tuples of data partition, *D*.

**Input:**

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

**Output:** A decision tree.

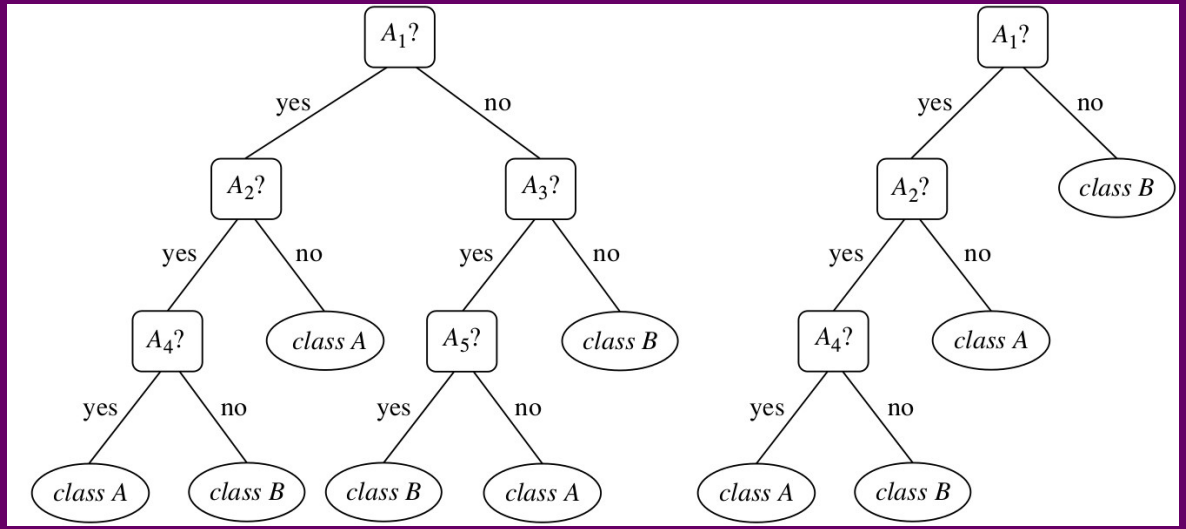
# ID3 Pseudo code

**Method:**

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C, then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12)     if D_j is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(D_j, attribute_list) to node N;
     endfor
(15) return N;
```

# Tree Pruning

- *Overfitting* problem: Some branches in the created decision tree may reflect noise or other anomalies in the data
  - Tree pruning removes least-reliable branches to increase overall quality of the tree



# Tree Pruning Approaches

- Pre-pruning
  - Halt tree construction to evaluate goodness of split.
  - If splitting results in “goodness” measure (e.g. information gain) dropping below predefined threshold, don't do split (make node a leaf)
    - Problem: Choosing good threshold value
- Post-pruning
  - Removes sub-trees from already constructed trees by replacing them with a leaf node (majority voting in subtree)

# Tree Pruning Approaches

- Cost-complexity pruning algorithm
  - Post-pruning approach
  - Tree cost-complexity function of size (number of leaves) and error-rate when classifying
  - Uses a pruning-set of data (different from training and test data sets) to generate a set of progressively pruned trees
  - Tree with smallest size and smallest cost-complexity is best

*K-Nearest Neighbor*

# K-Nearest Neighbor

- All tuples correspond to a position in N-dimensional space
  - N = number of attributes
- Main idea: When presented with a training (or unseen) tuple look at the K nearest neighbors and use their value for classification/prediction
  - Discrete-value classification: Majority voting among K neighbors
  - Continuous-value prediction: Return mean value of K neighbors

# K-Nearest Neighbor

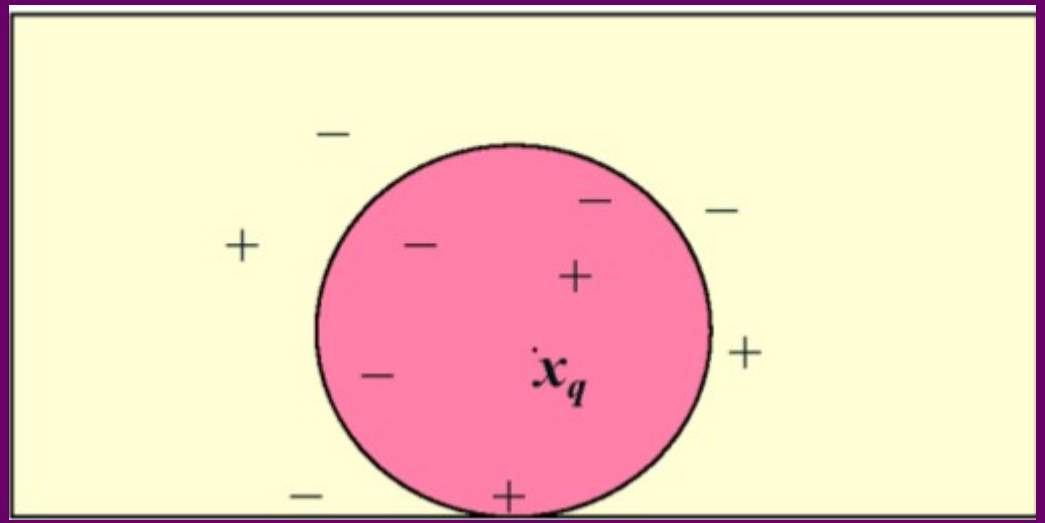
- "Nearness" found by using a distance metric like Euclidian distance (good idea to use normalized data)

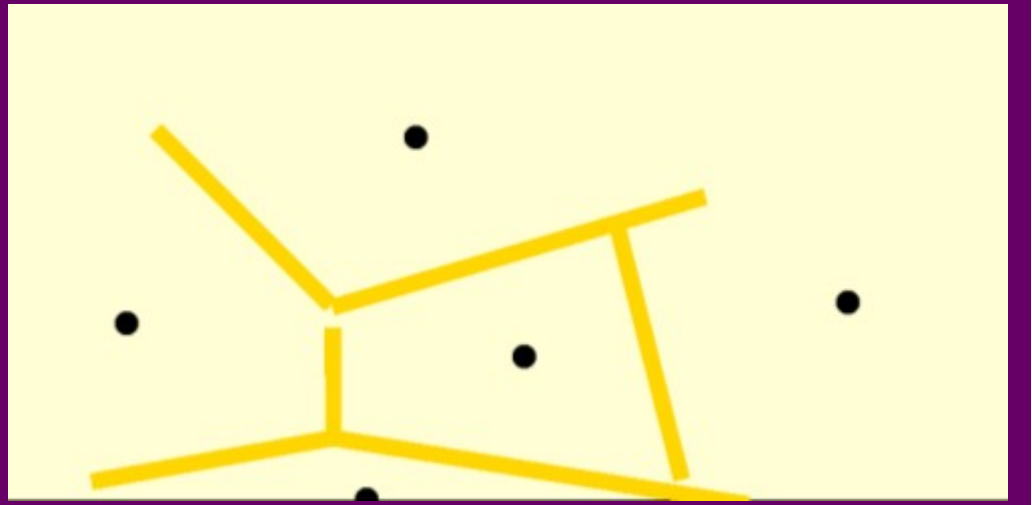
The most popular distance measure is **Euclidean distance** (i.e., straight line or "as the crow flies"). Let $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \ldots, x_{jp})$ be two objects described by $p$ numeric attributes. The Euclidean distance between objects $i$ and $j$ is defined as

$$d(i, j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}. \quad (2.16)$$

- What about nominal attributes?
  - Simple method is to simply compare values, if same value distance is 0 for that attribute, otherwise 1

# K-Nearest Neighbor



# K-Nearest Neighbor

- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the K neighbors according to their distance to the training/unseen tuple Xq
    - Give greater weight to closer neighborss

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

- Robust to noisy data by averaging K nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
- Finding best k requires doing experiments, for instance looking at error rate when using different values for k

*Evaluating Classification Models*

# Terminology

- Negative/Positive tuple
  - Positive tuple: tuples belonging to the class of most interest
  - Negative tuple: all other tuples
- Four essential values when measuring performance
  - True Positives (TP): The number of positive tuples that were correctly classified
  - True Negatives (TN): The number of negative tuples that were correctly classified
  - False Positives (FP): The number of negative tuples that were incorrectly classified as positive
  - False Negatives (FN): The number of positive tuples that were incorrectly classified as negative
- TP and TN tells us when the classifier is doing things right, and FP/FN tells us the opposite

# Confusion Matrix

- The terms TP, TN, FP and FN can be summarized in a confusion matrix

| | | Predicted class | | |
|---|---|---|---|---|
| | | *yes* | *no* | Total |
| **Actual class** | *yes* | *TP* | *FN* | *P* |
| | *no* | *FP* | *TN* | *N* |
| | Total | $P'$ | $N'$ | $P + N$ |

# Evaluation Measures

| *Measure* | *Formula* |
|---|---|
| accuracy, recognition rate | $\frac{TP+TN}{P+N}$ |
| error rate, misclassification rate | $\frac{FP+FN}{P+N}$ |
| sensitivity, true positive rate, recall | $\frac{TP}{P}$ |
| specificity, true negative rate | $\frac{TN}{N}$ |
| precision | $\frac{TP}{TP+FP}$ |
| $F$, $F_1$, $F$-score, harmonic mean of precision and recall | $\frac{2 \times precision \times recall}{precision + recall}$ |
| $F_\beta$, where $\beta$ is a non-negative real number | $\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$ |

# Evaluation Measures

- Accuracy = percentage of test tuples that are correctly classified
  - Is most effective when class label distribution is relatively balanced

$$accuracy = \frac{TP + TN}{P + N}.$$

# Evaluation Measures

- Error rate = percentage of test tuples that are incorrectly classified

$$error\ rate = \frac{FP + FN}{P + N}.$$

# Class Imbalance Problem

- When class label distribution is unbalanced and the main class label of interest is rare
  - I.e. there is a significant majority of the negative class and minority positive class
- In this case accuracy may be misleading
- Therefore we use sensitivity (proportion of positive tuples correctly classified) and specificity (proportion of negative tuples correctly classified) to measure classifier performance

# Evaluating Accuracy

- Holdout method
  - Data is randomly partitioned into two independent sets
    - Training set (e.g. 2/3) for classifier construction
    - Test set (e.g. 1/3) for accuracy estimation
  - Random subsampling (variation of holdout method)
    - Repeat holdout k times, accuracy = average of estimated accuracies

# Evaluating Accuracy

- K-fold cross validation
  - Randomly partition the data into k mutually exclusive subsets $\{D_1, D_2, \ldots D_k\}$, each approximately of the same size
  - At ith-iteration, $D_i$ as test set and others as training set
  - Leave-One-Out: k-fold cross validation when k=number of tuples in data set, usually used for small data set
  - Stratified cross-validation: each subset or “fold” is stratified to ensure that class label distribution is approximately the same as in initial data (i.e. before making the subsets)
  - Stratified 10-fold cross validation most popular

# Comparing Classification/Prediction Models

- Imagine that you have constructed two models to classify/predict an attribute – how do you know which model is best?
- Intuitive to select model with highest accuracy – but our accuracy measures are just an estimate.
- How do we really know if one model is better than the other?
- We need to employ a *test of statistical significance* to determine whether or not the two models are significantly different from one another
  - Example: t-test / students t-test

# Today's Lab

- Implement K-NN and/or ID3 using simple mushroom example data set
- We all meet in 4A58 to see if we can fit in
  - If not, we also use 4A54

*Thanks for listening!*

Send questions/feedback to andershh@itu.dk or use the Q&A forum on learnIT

*Bonus Slide from 2015: Predictor Error Measures*

# Predictor Error Measures

- Measure predictor accuracy: measure how far off the predicted value is from the actual known value
- **Loss function**: measures the error betw. $y_i$ and the predicted value $y_i'$
  - Absolute error: $| y_i - y_i' |$
  - Squared error: $(y_i - y_i')^2$
- Test error (generalisation error): the average loss over the test set
  - Mean absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{d}$ Mean squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{d}$
  - Relative absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{\sum_{i=1}^{d} | y_i - \bar{y} |}$ Relative squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{\sum_{i=1}^{d} (y_i - \bar{y})^2}$

  The mean squared-error exaggerates the presence of outliers

  Popularly use (square) root mean-square error, similarly, root relative squared error