Decision Trees

Based on C4.5 algorithm of Quinlan

By Janett Niendorf

Seminar: Classical Topics in Machine Learning, WS 2010/2011
At Technical University Berlin
> Department of Software Engineering and Theoretical Computer Science
> Machine Learning
(held 11.2.2011)
Overview

1 Basics Concept
   1 Supervised Learning with decision trees
   2 Training data and test data
   3 Parts of a decision tree

2 Five questions of tree building
   1 Number of branches
   2 Attribute selection
   3 Numeric data
   4 Pruning
   5 Missing attributes

3 C4.5 algorithm

4 Complexity

5 Classifier evaluation
Supervised Learning /or Classification/ or Inductive Learning

- Analogous to human learning based on experience
- For a “computer without experiences” this means: learn from data which represents the past

Decision tree:

- What features make an individual prone to sunburn?
  - ‘Sunburn’ ↔ ‘No sunburn’
- Will a person pay back a credit?
  - ‘Yes’ ↔ ‘No’
Collected data

- Each data record describes a piece of “past experience”

\[ A \cap C = \emptyset \]

### Attribute \((A)\)

<table>
<thead>
<tr>
<th>Age</th>
<th>Has_job</th>
<th>Own_house</th>
<th>Credit_rating</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>young</td>
<td>false</td>
<td>false</td>
<td>fair</td>
<td>No</td>
</tr>
<tr>
<td>young</td>
<td>false</td>
<td>false</td>
<td>good</td>
<td>No</td>
</tr>
<tr>
<td>middle</td>
<td>false</td>
<td>false</td>
<td>fair</td>
<td>No</td>
</tr>
<tr>
<td>middle</td>
<td>true</td>
<td>true</td>
<td>good</td>
<td>Yes</td>
</tr>
<tr>
<td>middle</td>
<td>false</td>
<td>true</td>
<td>excellent</td>
<td>Yes</td>
</tr>
<tr>
<td>old</td>
<td>false</td>
<td>true</td>
<td>excellent</td>
<td>Yes</td>
</tr>
<tr>
<td>old</td>
<td>false</td>
<td>false</td>
<td>fair</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

→ Predict classes of new cases

Reference [2]
Training & Test set

- Training set: $D_{\text{train}}$
- Test set (or holdout set): $D_{\text{test}}$

$$D = D_{\text{train}} \cup D_{\text{test}}$$

Note: all examples in $D$ are already labeled with a class (supervised learning)
Learning Process

Step 1: Training

- Training data $D_{train}$
- Learning algorithm

Step 2: Testing

- Model
- Test data $D_{test}$

Accuracy
Parts of a decision tree

- **Internal nodes incl. root node** = **decision nodes** = attributes
  ...specifies a test (i.e. ask a question)

- **Branches** = **outcomes** = possible values of an attribute
  ...answers towards a question

- **Leaf node** = **classification node** = a class
  ...the prediction

Decision Trees
Example decision tree

<table>
<thead>
<tr>
<th>Age</th>
<th>Has_job</th>
<th>Own_house</th>
<th>Credit_rating</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>young</td>
<td>false</td>
<td>false</td>
<td>good</td>
<td></td>
</tr>
</tbody>
</table>
Best solution

- We don't care about credit-rating (German Schufa) or age, instead we will give a credit to people who own a house or have job, cause we learned that those people are likely to pay back their debts.
Questions towards decision trees

1. How many decision outcomes or splits will there be at a node? Restriction to binary or multiple values of attributes?

2. Which property, attribute, should be tested at a node?

3. How should numeric attributes be handled?

4. If the tree becomes “too large”, how can it be made smaller and simpler, i.e., pruned?

5. How should missing attributes be handled?
Branching factor

1. How many decision outcomes or splits will there be at a node? Restriction to binary or multiple values of attributes?
   - Question of the designer
     - CART (Classification and Regression Tree): binary → deep
     - C4.5: multiple → wide trees
   - every tree can be transformed in a binary tree (branching factor=2), using just binary decision (yes/no answers)
   - “Mission Impossible” for attributes like id, is unique
Which attribute should be tested at a node?

- Idea: measure how the data could be split into the most pure parts

- Impurity functions
  - C4.5: entropy $\rightarrow$ information gain $\rightarrow$ information gain ratio
  - CART: gini impurity
**Entropy function** (dt.: Informationsverhältnis)

- Impurity reduction based on entropy (information theory),
  - $0$ ..means being pure, just one class
  - $1$ ..impure to the most extent

\[
\text{entropy}(D) = -\sum_{j=1}^{\mid C \mid} P(c_j) \log_2 P(c_j)
\]

- Example:
  - D has 50% positive and 50% negative instances
    \[
    \text{entropy}(D) = -0,5 \cdot \log_2 0,5 - 0,5 \cdot \log_2 0,5 = 1
    \]
  - D has 100% positive and 0% negative instances
    \[
    \text{entropy}(D) = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 = 0
    \]
    - Per definition $0 \cdot \log 0 = 0$
**Information gain** (dt.: Informationsgewinn)

- measures the reduction in impurity/disorder of attribute $A_i$:

\[
gain(D, A_i) = \text{entropy}(D) - \sum_{j=1}^{v} \frac{|D_j|}{|D|} \cdot \text{entropy}(D_j)
\]

\[
\text{entropy}_{A_i}(D)
\]

- Attribute with max. gain is selected

- disadvantage:
  - Attributes with many values are preferred
Information gain ratio (dt.: Informationsgewinn Verhältnis)

- Reduces disadvantage of information gain

\[
\text{gainRatio}(D, A_i) = \frac{\text{gain}(D, A_i)}{s \sum_{j=1}^{s} \left( \frac{|D_j|}{|D|} \cdot \log_2 \frac{|D_j|}{|D|} \right)}
\]

s: Number of possible values of attribute \( A_i \)

If \( A_i \) has many values, the denominator grows and gain ratio decreases.
Continuous attributes (numeric)

3 How should continuous attributes be handled? (zip-codes, boolean attributes, counts vs. price or temperature)

A partition of the data space in intervals axis-parallel! → Decision tree.
Continuous attributes problem and solution

• Discretize numeric attributes
  - Sort instances according to attribute’s values
  - Place breakpoints where the class changes

• Problem of overfitting
  - procedure is very sensitive to noise
  - one instance with an incorrect class label will probably produce a separate interval

• Simple solution:
  - enforce minimum number of instances of a class per interval
Pruning

If the tree becomes “too large”, how can it be made smaller and simpler?

- Pruning reduces Overfitting:
  - Pre-Pruning: stopped splitting while tree building
  - Post-Pruning: with the fully-grown decision tree

Partition with pruning of noise. → Decision Tree.
Pre-pruning vs. Post-pruning

Pre-pruning:
- Obviously dangerous, can “stop too early”, horizon effect
- Post-pruning preferred in practice

• Post-pruning
  - Fully-grown tree shows all attribute interactions
  - Two pruning operations:
    » 1. Sub-tree replacement by a leaf \( O(|D|) \)
    » 2. Sub-tree raising \( O(|D| (\log |D|)^2) \)
      \( \rightarrow \) because of redistribution of instances
Expected error pruning

- Idea: sub-tree is pruned if the estimated error for a leaf is less than the error of the sub-tree
  - Has low confidence with training data
- Use of a separate validation set, then ...
- Available data sets:

  - After tree is built, classification of validation set
    - Find errors at each node
    - Pruning based on the errors
Missing attribute values

1. How should missing attributes be handled?

<table>
<thead>
<tr>
<th>Age</th>
<th>Has_job</th>
<th>Own_house</th>
<th>Credit_rating</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>middle</td>
<td>true</td>
<td>true</td>
<td>good</td>
<td>Yes</td>
</tr>
<tr>
<td>old</td>
<td></td>
<td>true</td>
<td>excellent</td>
<td>Yes</td>
</tr>
<tr>
<td>old</td>
<td>false</td>
<td>false</td>
<td>fair</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Do not use those data sets - wasteful, just if many records
- Simple idea: treat missing ones as a separate value “unknown” (also used in tree)
- the most frequent value (discrete)/ mean (continuous)
- If not appropriate (example Man.pregnant=true), advanced method ...

Decision Trees
Missing attribute values advanced

C4.5

• Distribution proportionally according to the distribution of attributes with values
  - A piece going down a branch receives a weight proportional to the popularity of the branch
  - weights sum to 1
    » 20% missing, 30% true, 50% false
      → records with true provide 3/8
      → records with false provide 5/8

• Information gain works with fractional instances
  - use sums of weights instead of counts
Learning algorithm C4.5 (for discrete values)

\textbf{decisionTree}(D, A, T)

\begin{itemize}
\item $D$ contains only training examples of the same class?
\item Make $T$ a leaf node of the most frequent class in $D$
\item Calculate for every attribute in $A$ the impurity for the resulting partition and select attribute $A_g$ with the max. impurity reduction
\item [optional: pre-pruning]
\item max. impurity reduction with attribute $A_g < \text{threshold}$
\item Make $T$ a decision node on $A_g$
\item Partition $D$ in $m$ disjoint subsets based on the $m$ values of attribute $A_g$
\item For every non-empty partition $D_j$ in $\{D_1, D_2, ..., D_m\}$ create a child node $T_j$ and call recursively: $\text{decisionTree}(D_j, A-\{A_g\}, T_j)$
\end{itemize}
Complexity

- Decision trees generalize data
  - data transformed into a tree gets more smaller/more compact description of the data
- constructing an optimal binary decision tree is known to be NP-complete (1976, Hyafil, Rivest; [3])
- optimal tree: "minimizes the expected number of tests required to identify the unknown object"
- NP-complete means
  - A classification can be given in polynomial time, that is the height of the tree $O(\log n)$
  - But finding the optimal tree is of exponential nature, such that with a high input, more possibilities are to explore than computers on the planet
Heuristics

- Because of complexity:
  - all decision tree algorithms use heuristic methods
- greedy algorithm:
  - impurity finds locally optimal decisions at each node
  - hope of finding the global optimum
  - after each split: divide-and-conquer

- Time for tree building in C4.5:
  - Discrete values: $O(|A|*|D|)$
    ...grows linearly with size of data set $D$ per attribute in $A$
  - Continuous values: $O(|A|*|D|\log|D|)$
    ...because sorting is needed
Classifier Evaluation

Training data $D_{\text{train}}$ → Learning algorithm → Model → Test data $D_{\text{test}}$ → Accuracy

Accuracy = \( \frac{\text{Number of correct classifications}}{\text{Total number of test classes}} \)
Partition Approaches

- Choose earlier collected data for the training set and latest data for testing (reflect dynamic aspect of applications)
- Random samples for training, rest for testing
- Multiple Random Sampling
  - Perform random sampling n-times
  - Each time a different training set,
  - Produces n measurements, final is average
Cross-Validation

- **n-fold Cross-Validation**
  - Every partition is used once as a test case \( (D_{\text{test}}) \), whereas the other \( n-1 \) parts built the model together \( (D_{\text{train}}) \)
  - \( n \)-times repeated, average of \( n \)-results is final result
  - Advantage of this method over repeated random sub-sampling: observations are used for both training and testing

- **Leave-one-out Cross-Validation**
  - Special-case of \( n \)-fold Cross Validation, where: \( n \) ...is the number of examples in the available data, used when small data set given
Confusion Matrix

Classification:

- positive
- negative

True-Positive
False-Negative

False-Positive
True-Negative

Decision Tree Algorithm

Test data
Training data
Accuracy and other measures

- which measure you use, depends on your problem

\[
\text{accuracy} = \frac{FP + FN}{TP + TN + FP + FN} = \frac{\text{Number of correct classifications}}{\text{Total number of test classes}}
\]

\[
\text{error rate} = 1 - \text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]

\[
\text{Precision} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}}
\]

\[
\text{Recall} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}}
\]
Still time?

- ... lets have a look at the Weka data mining tool:
  
  http://www.cs.waikato.ac.nz/ml/weka/
Reference List


