Decision Trees/Rules
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
Decision Trees

1. Introduction
   - Algorithm
   - Example
   - Information gain bias
   - Special Data
   - Overfitting/Pruning
   - Limitations/Other Algorithms

2. Rule induction
   - Sequential covering algorithms
   - Inductive Logic Programming
We can approach learning a concept as the algorithm to find the set of questions that is needed to distinguish it from others.

We can use a tree of questions as a representation language, each node from the tree is a test about an attribute.

This representation is equivalent to a DNF ($2^n$ concepts).

To learn we have to perform a search in the space of trees of questions.
Decision Trees

- Searching in the space of all DNF is too costly.
- To reduce the computational cost we impose a bias (the concepts that are preferred).
- **Constraint:** We want the tree that represents the minimal description of the target concept given a set of examples.
- **Justification:** This tree will be the best for classifying new examples (the probability of it having unnecessary questions is reduced).
- **Occam’s razor:** “Plurality ought never be posited without necessity”
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
The first algorithm for decision trees was **ID3** (Quinlan 1986)

It is a member of the family of algorithms for Top Down Induction Decision Trees (TDIDT)

ID3 performs a Hill-Climbing search in the space of trees

For each new question an attribute is chosen and the examples are partitioned accordingly to their values, this procedure is repeated recursively with each partition until all examples are from the same concept

The selection of the attribute is decided using an heuristic function that bias the selection towards the minimal tree
Decision Trees Algorithms
Information Theory

- Information theory studies (among other things) how to code messages and the cost of their transmission.
- Given a set of messages $M = \{m_1, m_2, \ldots, m_n\}$, each one with probability $P(m_i)$, we can define the quantity of information ($I$) contained in the messages $M$ as:

$$I(M) = \sum_{i=1}^{n} -P(m_i) \log(P(m_i))$$

- This value can be interpreted as the information necessary to distinguish among the messages from $M$ (number of bits necessary to code the messages).
We can establish an analogy between learning and message coding assuming that the classes are the messages and the proportion of examples from each class are their probability.

A decision tree can be seen as the coding that allows to distinguish among classes.

(Learn a decision tree $\iff$ Learn a code)

We are looking for the minimal coding.

Each attribute must be evaluated to decide if it is included in the code.
Quantity of information as classification heuristic

- We are implicitly minimizing the error \( E_{emp} \)
  - For this heuristic an attribute is better if it can distinguish better among classes (0/1 Loss)
- We are implicitly looking for the smallest hypothesis
  - We are minimizing the size of the hypothesis using local information (quantity of information of an attribute)
  - We assume that reducing locally the number of bits of the code will minimize the global number of bits
Quantity of information as classification heuristic

- At each level of the tree we look for the attribute that allows to minimize the code.
- This attribute is the one that minimizes the remaining quantity of information (bits that are left to code).
- The selection of the attribute must result in a partition of examples where each subset is biased towards one of the classes.
- We need an heuristic that measures the remaining information quantity induced by an attribute (Entropy, $E$).
Quantity of information

Given a set of examples $\mathcal{X}$ and being $\mathcal{C}$ their classification

$$I(\mathcal{X}, \mathcal{C}) = \sum_{\forall c_i \in \mathcal{C}} - \frac{\#c_i}{\#\mathcal{X}} \log(\frac{\#c_i}{\#\mathcal{X}})$$

Bits needed to code the examples without additional information
Entropy

- Given an attribute $A$ and being $[A(x) = v_i]$ the examples with value $v_i$ for the attribute

$$E(\mathcal{X}, A, C) = \sum_{\forall v_i \in A} \frac{\#[A(x) = v_i]}{\#\mathcal{X}} \cdot I([A(x) = v_i], C)$$

- Bits needed to code the examples given an attribute
  (Just the weighted sum of the information of the induced partition)
Information gain

**Bits that remain to be coded**

\[ G(\mathcal{X}, A, C) = I(\mathcal{X}, C) - E(\mathcal{X}, A, C) \]

\[
\begin{array}{c}
\text{C1} & \text{C2} & \text{C3} \\
A=v1 & A=v2 & A=v3
\end{array}
\]

\[
\begin{array}{c}
\text{C1} & \text{C2} & \text{C3} \\
A=v1 & A=v2 & A=v3
\end{array}
\]

\[
\begin{array}{c}
\text{C1} & \text{C2} & \text{C3} \\
A=v1 & A=v2 & A=v3
\end{array}
\]
ID3 algorithm

**Algorithm:** ID3 ($\mathcal{X}$: Examples, $\mathcal{C}$: Classification, $\mathcal{A}$: Attributes)

if all examples are from the same class

then

return a leave with the class name

else

Compute the quantity of information of the examples ($I$)

foreach attribute in $\mathcal{A}$ do

Compute the entropy ($E$) and the information gain ($G$)

end

Pick the attribute that maximizes $G(a)$

Delete $a$ from the list of attributes ($\mathcal{A}$)

Generate a root node for the attribute $a$

foreach partition generated by the values of the attribute $a$ do

Tree$_i$=ID3(examples from $\mathcal{X}$ with $a=v_i$, examples classification, rest of attributes)

generate a new branch with $a=v_i$ and Tree$_i$

end

return the root node for $a$

end
Algorithm ID3 - special cases

In practical cases we will encounter some problems:

- The information gain is very small for all the attributes
  - It means that the remaining attributes are not discriminant
  - To continue adding decisions to the tree means to include arbitrary decisions (and increase the size of the tree)
  - The best action is to stop and classify all the examples in the majority class

- A value from the attribute has no examples
  - Classify the value as the majority class

- Examples of different classes in a terminal node
  - Classify the value as the majority class
Decision Trees

- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

Rule induction

- Sequential covering algorithms
- Inductive Logic Programming
Example (1)

Let's use the following set of examples

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Eyes</th>
<th>Hair</th>
<th>Height</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Blue</td>
<td>Blond</td>
<td>Tall</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
<td>Black</td>
<td>Medium</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>Brown</td>
<td>Black</td>
<td>Medium</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Green</td>
<td>Black</td>
<td>Medium</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Green</td>
<td>Black</td>
<td>Tall</td>
<td>+</td>
</tr>
<tr>
<td>6</td>
<td>Brown</td>
<td>Black</td>
<td>Short</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Green</td>
<td>Blond</td>
<td>Short</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>Blue</td>
<td>Black</td>
<td>Medium</td>
<td>+</td>
</tr>
</tbody>
</table>
Example (2)

\[
I(X, C) = -\frac{1}{2} \log(\frac{1}{2}) - \frac{1}{2} \log(\frac{1}{2}) = 1
\]

\[
E(X, \text{eyes}) = (\text{blue}) \frac{3}{8} \left( -1 \cdot \log(1) - 0 \cdot \log(0) \right)
+ (\text{brown}) \frac{2}{8} \left( -1 \cdot \log(1) - 0 \cdot \log(0) \right)
+ (\text{green}) \frac{3}{8} \left( -\frac{1}{3} \log(\frac{1}{3}) - \frac{2}{3} \log(\frac{2}{3}) \right)
= 0.344
\]

\[
E(X, \text{hair}) = (\text{blond}) \frac{2}{8} \left( -\frac{1}{2} \cdot \log(\frac{1}{2}) - \frac{1}{2} \cdot \log(\frac{1}{2}) \right)
+ (\text{black}) \frac{6}{8} \left( -\frac{1}{2} \cdot \log(\frac{1}{2}) - \frac{1}{2} \cdot \log(\frac{1}{2}) \right)
= 1
\]

\[
E(X, \text{height}) = (\text{tall}) \frac{2}{8} \left( -\log(1) - 0 \cdot \log(0) \right)
+ (\text{medium}) \frac{4}{8} \left( -\frac{1}{2} \cdot \log(\frac{1}{2}) - \frac{1}{2} \cdot \log(\frac{1}{2}) \right)
+ (\text{short}) \frac{2}{8} \left( 0 \cdot \log(0) - 1 \cdot \log(1) \right)
= 0.5
\]
As we can see the attribute **eyes** is the one that maximizes the information gain

\[
G(X, \text{eyes}) = 1 - 0.344 = 0.656 \\
G(X, \text{hair}) = 1 - 1 = 0 \\
G(X, \text{height}) = 1 - 0.5 = 0.5
\]
Example (4)

This attributes induces a partition that forms the first level of the tree

```
EYES
  +
  BLUE
  1,2,8
  +
  BROWN
  3,6
  −
  GREEN
  4,7
  −
  5
  +
```
Now only in the node corresponding to the value \textbf{green} we have a mixture of examples from the two classes, we repeat the process with these examples.

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Ej. & Hair & Height & Class \\
\hline
4 & Black & Medium & - \\
5 & Black & Tall & + \\
7 & Blond & Short & - \\
\hline
\end{tabular}
\end{center}
Example (6)

\[ I(X, C) = -\frac{1}{3} \cdot \log(1/3) - \frac{2}{3} \cdot \log(2/3) = 0.918 \]
\[ E(X, \text{hair}) = (\text{blond}) \frac{1}{3} \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \]
\[ + (\text{black}) \frac{2}{3} \cdot (-\frac{1}{2} \cdot \log(1/2) - \frac{1}{2} \cdot \log(1/2)) \]
\[ = 0.666 \]
\[ E(X, \text{height}) = (\text{tall}) \frac{1}{3} \cdot (0 \log(0) - 1 \cdot \log(1)) \]
\[ + (\text{medium}) \frac{1}{3} \cdot (-1 \cdot \log(1) - 0 \cdot \log(0)) \]
\[ + (\text{short}) \frac{1}{3} \cdot (0 \cdot \log(0) - 1 \cdot \log(1)) \]
\[ = 0 \]
Now the attribute that maximizes the information gain is \textit{eyes}.

\[
G(X, \text{hair}) = 0.918 - 0.666 = 0.252
\]
\[
G(X, \text{height}) = 0.918 - 0 = 0.918^*
\]
Now the tree can discriminate all the examples
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
The information gain measure is biased towards the attributes that have more values

**Problem:** If an attribute has a large number of values probably the resulting tree will be larger.

The reason for that bias resides in the weight given to the values that are totally discriminant.
Shortcomings of the information gain measure - Example

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>n. exam</th>
<th></th>
<th>$A_2$</th>
<th>n. ejem</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20+</td>
<td></td>
<td>a</td>
<td>40+</td>
</tr>
<tr>
<td>b</td>
<td>25+</td>
<td></td>
<td>b</td>
<td>40-</td>
</tr>
<tr>
<td>c</td>
<td>20-</td>
<td></td>
<td>c</td>
<td>10+ 10-</td>
</tr>
<tr>
<td>d</td>
<td>15-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>5+ 15-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$G(A_1) = 1 - \left( \frac{a}{5} \cdot 0 + \frac{b}{4} \cdot 0 + \frac{c}{5} \cdot 0 + \frac{d}{20} \cdot 0 + \frac{e}{5} \cdot \left( -\frac{1}{4} \cdot \log \frac{1}{4} \right) - \frac{3}{4} \cdot \log \frac{3}{4} \right) = 0.84$

$G(A_2) = 1 - \left( \frac{a}{5} \cdot 0 + \frac{b}{5} \cdot 0 + \frac{c}{5} \cdot \left( -\frac{1}{2} \cdot \log \frac{1}{2} \right) - \frac{1}{2} \cdot \log \frac{1}{2} \right) = 0.8$
A normalized value

- This problem can be reduced by normalizing the information gain with another function that measures the distribution of the examples with the values of an attribute

\[
SI(\mathcal{X}, A) = - \sum_{\forall v_i \in A} \frac{\#[A(x) = v_i]}{\#\mathcal{X}} \cdot \log\left( \frac{\#[A(x) = v_i]}{\#\mathcal{X}} \right)
\]

- In the example:

\[
\frac{G(A_1)}{SI(A_1)} = \frac{0.86}{2.3} = 0.36 \quad \frac{G(A_2)}{SI(A_2)} = \frac{0.8}{1.52} = 0.52
\]
1. Decision Trees
   - Introduction
   - Algorithm
   - Example
   - Information gain bias
   - Special Data
     - Overfitting/Pruning
     - Limitations/Other Algorithms

2. Rule induction
   - Sequential covering algorithms
   - Inductive Logic Programming
Numerical Attributes

- The defined heuristic functions only work for discrete attributes.
- Usually data sets are also described by numerical attributes.
- To treat this attributes as discrete attributes is unfeasible (each value as a different value).
- **Solution:** To partition the range of values in two or more intervals.
- In practice this means to discretize the attribute (dynamically).
Discretization of numerical attributes

- We begin with the ordered sequence of values $A_i = \{v_1, v_2, ..., v_n\}$
- The information gain is measured for each partition of the sequence ($n - 1$ sets of binary partitions)
- The partition that maximizes the information gain is then compared with the rest of the attributes
- The decision used for the node of the tree is $[A[x] < v_i]$ and $[A[x] \geq v_i]$
- We can not discard the attribute from successive decision (we can partition further in smaller intervals)
- This will increment the computational cost of each decision
Numerical attributes - Node decisions

A < n

A >= n

A >= n y A < m

A >= m
## Discretization of numerical attributes

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>.89</td>
<td>.76</td>
<td>.6</td>
<td>.39</td>
<td>0</td>
<td>.39</td>
<td>.6</td>
<td>.76</td>
<td>.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>.89</td>
<td>.76</td>
<td>.6</td>
<td>.39</td>
<td>.72</td>
<td>.39</td>
<td>.6</td>
<td>.76</td>
<td>.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>.89</td>
<td>.76</td>
<td>.96</td>
<td>1</td>
<td>.97</td>
<td>1</td>
<td>.96</td>
<td>.76</td>
<td>.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.82</td>
<td>.76</td>
<td>.87</td>
<td>.87</td>
<td>.84</td>
<td>.79</td>
<td>.68</td>
<td>.84</td>
<td>.82</td>
<td></td>
</tr>
</tbody>
</table>
A **missing value** is a value that is unknown for an attribute of an example.

In practice real data sets have this kind of values.

We can find problems processing these values when:

- The best attribute is picked (how to compute the information gain)
- The branches of an attribute are created (where to put the examples)
- Predicting the class of a new example (what branch to follow)
Missing values - Solutions

When picking the best attribute

1. Ignore the examples
2. Scale the information gain

\[
G'(\mathcal{X}, A, C) = \frac{\#[A(x) \neq v_{\text{miss}}]}{\#\mathcal{X}} \cdot G(\mathcal{X}, A, C)
\]

3. Substitute the values (mode / median)
Missing values - Solutions

Creating the branches

1. Ignore the examples
2. Use the most frequent value
3. Assign a fraction of the example to each branch proportionally to the distribution of the values of the attribute.
4. Include the instance on each branch
5. Add a new branch for the missing values
Missing values - Solutions

Predicting a new example

1. Use the branch for missing values if there is one
2. Use the most frequent value
3. Explore all the branches and return a probability for each class
4. Return the majority class
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
Overcoming noise

- **Noise**: Examples that are wrongly classified
- It is usual for real datasets that a part of the examples are misclassified
- Our model size increases to minimize the error (overfitting)
- Extra nodes appear to distinguish the misclassified examples
- Overfitting reduces generality and as consequence the prediction accuracy is reduced
Pruning of decision trees

- A way to reduce the effect of noise is to delete the subtrees that incorrectly classify examples.
- These techniques are called **pruning**.
- The idea is to find the nodes that will improve the global accuracy if deleted from the tree.
- Two possibilities:
  - **Pre-pruning**: prune while the tree is built
  - **Post-pruning**: prune after the tree is completed
Pre-pruning

- It is based on statistical tests that determine the benefits of creating a new decision.
- The accuracy of the best decision for a node is compared with the accuracy obtained by stopping creating new decisions.
- Test for comparing probability distributions:

\[
\sum_{c_i \in C} \sum_{v_i \in A} \frac{(P(c_i | A(x) = v_i) - P(c_i))^2}{P(c_i)}
\]

distributes as a $\chi^2$ with $(c - 1) \cdot (v - 1)$ degrees of freedom.
Post-pruning

- The prediction error can be estimated using the examples from the dataset.
- Being \( N \) the number of examples in a node, \( E \) the number of examples that are not in the majority class and \( B \) the binomial probability distribution function, the estimated error for a node can be computed as:

\[
\text{Error} = N \cdot B_{cf}(E, N)
\]

- Where \( cf \) is the confidence factor of the binomial probability function.
- If the error of the node is less than the weighted sum of the errors of its descendants then they are pruned.
### Post-pruning - Example (I)

<table>
<thead>
<tr>
<th>Example</th>
<th>Price</th>
<th>Size</th>
<th>Country</th>
<th>Motor</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>expensive</td>
<td>big</td>
<td>france</td>
<td>diesel</td>
<td>compact</td>
</tr>
<tr>
<td>2</td>
<td>cheap</td>
<td>medium</td>
<td>germany</td>
<td>diesel</td>
<td>compact</td>
</tr>
<tr>
<td>3</td>
<td>medium</td>
<td>medium</td>
<td>germany</td>
<td>gasoline</td>
<td>compact</td>
</tr>
<tr>
<td>4</td>
<td>cheap</td>
<td>big</td>
<td>france</td>
<td>gasoline</td>
<td>compact</td>
</tr>
<tr>
<td>5</td>
<td>cheap</td>
<td>big</td>
<td>france</td>
<td>gasoline</td>
<td>compact</td>
</tr>
<tr>
<td>6</td>
<td>expensive</td>
<td>small</td>
<td>japan</td>
<td>gasoline</td>
<td>sports</td>
</tr>
<tr>
<td>7</td>
<td>expensive</td>
<td>small</td>
<td>germany</td>
<td>gasoline</td>
<td>sports</td>
</tr>
<tr>
<td>8</td>
<td>expensive</td>
<td>small</td>
<td>USA</td>
<td>gasoline</td>
<td>sports</td>
</tr>
<tr>
<td>9</td>
<td>medium</td>
<td>small</td>
<td>japan</td>
<td>gasoline</td>
<td>sports</td>
</tr>
<tr>
<td>10</td>
<td>medium</td>
<td>small</td>
<td>germany</td>
<td>gasoline</td>
<td>sports</td>
</tr>
<tr>
<td>11</td>
<td>expensive</td>
<td>big</td>
<td>USA</td>
<td>gasoline</td>
<td>luxury</td>
</tr>
<tr>
<td>12</td>
<td>expensive</td>
<td>big</td>
<td>USA</td>
<td>diesel</td>
<td>luxury</td>
</tr>
<tr>
<td>13</td>
<td>expensive</td>
<td>big</td>
<td>france</td>
<td>gasoline</td>
<td>luxury</td>
</tr>
<tr>
<td>14</td>
<td>expensive</td>
<td>big</td>
<td>germany</td>
<td>gasoline</td>
<td>luxury</td>
</tr>
<tr>
<td>15</td>
<td>expensive</td>
<td>big</td>
<td>germany</td>
<td>diesel</td>
<td>luxury</td>
</tr>
</tbody>
</table>
Post-pruning - Example (2)

Decision Trees

Overfitting/Pruning

Javier Béjar (LSI - FIB)

Term 2012/2013
Post-pruning - Example (2)

Decision Trees/Rules

Size
- Medium: [2,3] (Compact)
- Big: [1,4,5], [11,12,13,14,15] (Compact, Luxury)
- Small: [6,7,8,9,10] (Sports)

Price
- Cheap: [4,5] (Compact)
- Expensive: [1], [11,12,13,14,15] (Compact, Luxury)
- Medium: ∅ (Luxury)
Post-pruning - Example (2)

Decision Trees/Rules

Term 2012/2013
Post-pruning - Example (2)
Post-pruning - Example (3)

Decision Tree:

```
Size
  | Medium (2,0,0) |
  | Big (3,0,5)   |
  | Small (0,5,0) |

Price
  | Cheap (2,0,0) |
  | Expensive (1,0,5) |
  | Medium ∅ |

Country
  | Japan ∅ |
  | USA (0,0,2) |
  | France (1,0,1) |
  | Germany (0,0,2) |
```
Post-pruning - Example (4)

- **Node Expensive**

  \[
  \text{Error}_{\text{expensive}} = 6 \cdot B_{0.25}(1, 6) = 6 \cdot 0.5339 = 3.203
  \]

- **Node Big**

  \[
  \text{Error}_{\{\text{USA, Ger, Fr}\}} = 2 \cdot B_{0.25}(0, 2) + 2 \cdot B_{0.25}(0, 2) + 2 \cdot B_{0.25}(1, 2) = 4.125
  \]

- **Node Big**

  \[
  \text{Error}_{\{\text{cheap, expensive}\}} = 2 \cdot B_{0.25}(0, 2) + 6 \cdot B_{0.25}(1, 6) = 4.327
  \]
Pruning / Noise and % accuracy

Noise vs Accuracy

- **No pruning**
- **Post pruning**
- **Reduced error**

Accuracy vs % Noise graph showing the relationship between noise percentage and accuracy for different pruning methods.
Pruning / Nose and tree size

Noise vs Size

- No pruning
- Post pruning
- Reduced error

Size vs % Noise graph showing the relationship between noise percentage and tree size for different pruning methods.
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
Limitations of ID3

- ID3 can learn any concept, but there are concepts that are more difficult to learn than others.
- The main problem is that the representation uses partitions that are parallel to the coordinate axis.
- To approximate this way some concepts could result in very complex trees.
Other algorithms for decision trees

- Different measures for choosing split attributes:
  - CART (Classification And Regression Trees)
    GINI index \( G(X, C) = 1 - \sum P(X|C)^2 \)

- Different conditions on nodes:
  - Oblique Classifiers: Linear combination of attributes

- Complex predictions schemes on leaves
  - Model trees: Decision trees where the leaves are a model obtained by other ML algorithm

- Prediction of continuous labels
  - Regression trees
Regression trees

- Sometimes we want to predict a function instead of a set of discrete values.
- There are several methods for approximating functions given a set of examples.
- Usually a specific global model is assumed (e.g., linear regression = linear function).
- Decision trees allow to divide the problem in several subproblems, each one can have a different model (or different instances of the same model).
Regression trees

\[ f(x) = ax + b \]

\[ f(x) = a_1 x + b_1 \]
\[ f(x) = a_2 x + b_2 \]
\[ f(x) = a_3 x + b_3 \]
Regression trees

- **Splitting function**: minimization of the sum of squared error

  \[
  sq\_error = \sum_{x}(f(x) - \hat{f}(x))^2
  \]

- **When to stop splitting?** ⇒ Penalization to the complexity of the model
  - Size of the leaves
  - Function of the size of the model

- **Model at the leaves** (computational cost!)
  - Mean value
  - Regression model (eg: linear regression)
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
Rule induction

- Decision rules can be generated from a decision tree.
- Each path from the root of the tree to a leaf is a rule that classifies a set of examples.
- It is also possible to generate a set of rules without creating a decision tree.
- These algorithms learn the rules sequentially and not all at once like in a decision tree.
- Some of these algorithms can even learn first order logic rules.
1 Decision Trees
   - Introduction
   - Algorithm
   - Example
   - Information gain bias
   - Special Data
   - Overfitting/Pruning
   - Limitations/Other Algorithms

2 Rule induction
   - Sequential covering algorithms
   - Inductive Logic Programming
Sequential covering algorithms

- These algorithms generate the rules sequentially by looking for the best rule that covers a subset of the examples.
- We look for the rule with the best accuracy but not the maximum coverage.
- Once we have the best rule, we can eliminate the examples covered.
- This procedure can be iterated until we have rules that cover all the dataset.
- The result is a disjunction of rules that can be ordered by accuracy.
Sequential covering algorithms - Algorithm

Algorithm: sequential-covering(attributes, examples, min)

list_rules = []
rule = learn-one-rule(attributes, examples)

while accuracy(rule, examples) > min do
    list_rules = list_rules + rule
    examples = examples - correct-examples(rule, examples)
    rule = learn-one-rule(attributes, examples)
end

list_rules = order-by-accuracy(list_rules)
Sequential covering algorithms

Rule induction
Sequential covering algorithms
Learning one rule

- The key point of these methods is how to learn the best rule given a set of examples.
- One possibility is to use the idea from decision trees and search in the space of conjunctions.
- We start with the empty rule and each step we select the best new conjunction for the rule using an heuristic (e.g., entropy) and a greedy strategy (e.g., keep the best).
- To avoid local optima, a more exhaustive search can be performed, for example using beam search and storing the $k$ best rules.
- The majority class of the examples selected is assigned as the rule prediction.
Learning one rule - Search example

Diagram of rule induction using sequential covering algorithms, showing the pruning process and accuracy metrics for different rule sets.

- Rule sets: [a=1], [a=2], [a=3]
- Branches:
  - [a=1] -> [a=1,b=1] -> [a=2,c=1,b=1] (Pruned)
  - [a=1] -> [a=1,b=2] (Pruned)
  - [a=2] -> [a=2,c=1] (Pruned)
  - [a=2] -> [a=2,c=2] (Pruned)
  - [a=3] -> [a=3,d=1] -> [a=3,d=1,c=1] (Acc=0.79)
  - [a=3] -> [a=3,d=2] (Pruned)

Accuracy metrics:
- Acc=0.76
- Acc=0.75
- Acc=0.79
Other algorithms for rule induction

- The sequential method has its limitations, there are a large variety of alternatives.
- For instance we can change how the instances are selected for covering (e.g., focusing on the classes).
- There are also other formalisms of rules that can be used:
  - Decision tables
  - Decision lists
  - Ripple down rules
  - Default rules
1 Decision Trees
- Introduction
- Algorithm
- Example
- Information gain bias
- Special Data
- Overfitting/Pruning
- Limitations/Other Algorithms

2 Rule induction
- Sequential covering algorithms
- Inductive Logic Programming
Inductive Logic Programming (ILP)

- Decision trees and rule algorithms can be seen as propositional rule learners
- These algorithms can be extended for learning first order rules (rules with variables)
- These algorithms allow to learn concepts in domains where knowledge can not be expressed as a set of attribute-value pairs
- This extends the complexity of the concepts that can be learned (relational knowledge, sub structures, programs, ...
Imagine the following set of examples:

<table>
<thead>
<tr>
<th>$number_1$</th>
<th>$number_2$</th>
<th>$number_3$</th>
<th>$larger_{12}$</th>
<th>$larger_{23}$</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>true</td>
<td>true</td>
<td>+</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
<td>true</td>
<td>true</td>
<td>+</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>true</td>
<td>true</td>
<td>+</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>2</td>
<td>false</td>
<td>true</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>true</td>
<td>false</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

and we want to learn the concept $number_1 >_{trans} number_3$

an induction tree learner will obtain rules like:

$$number_1 > 5 \land number_3 < 3 \land number_2 = 4 \rightarrow class = +$$

when we really want:

$$number_1 > number_2 \land number_2 > number_3 \rightarrow number_1 >_{trans} number_3$$
Inductive Logic Programming (ILP)

- We need to represent adequately the examples:
  \[\text{number}(1), \text{number}(2), \text{number}(3), \ldots\]
  \[\text{larger}(5,4), \text{larger}(4,3), \text{larger}(7,4), \ldots\]
  \[\neg \text{larger}(8,9), \neg \text{larger}(3,4), \ldots\]

- We need to represent the target predicate:
  \[\text{largetrans}(5,4,3), \text{largetrans}(7,4,1), \ldots\]
  \[\neg \text{largetrans}(8,9,2), \neg \text{largetrans}(5,3,4), \ldots\]

- We need a bias to reduce the search space (concepts we want to learn)

- Usually the target rules are *Horn clauses*

\[
H \lor \neg L_1 \lor \neg L_2 \cdots \lor \neg L_n \iff (L_1 \land \cdots \land L_n) \rightarrow H
\]
FOIL

- FOIL is an algorithm able to learn first order rules
- The idea is similar to the algorithms for rule learning and decision trees
  1. We begin with the most general rule ($\emptyset \rightarrow$ concept)
  2. Literals are generated and tested to specialize the rule
  3. The best candidate is chosen and added to the rule
  4. Examples covered by the rule are eliminated
FOIL

- **Candidate literals generation**
  - Positive or negative literals used in the examples with variables (including the goal), at least one variable from a previous literal or from the head of the rule
  - Equality or inequality literals using the variables from the literals
  - Arithmetic comparisons

- **Evaluating the literals/rules**
  - Several measures can be used (eg. Information gain, impurity)
  - Length of the rule (to avoid overfitting)
FOIL - example

We have the following examples:

father(john, mary), father(john, peter), father(paul, john),
father(paul, emma), father(harry, matt)
mother(emma, harry), mother(emma, james), mother(mary, jane)
grandfather(paul, peter), grandfather(john, jane), grandfather(paul, mary), grandfather(paul, harry), grandfather(paul, james)

The goal concept is grandfather

We assume negation by failure (other possible combination of constants and grandfather are false) (9 constants → 81 combinations, 5 are positive examples, 76 are negative)
FOIL - example

- We begin with the most general rule $\blacksquare \rightarrow \text{grandfather}(x, y)$
- We have several candidate clauses (we only evaluate a subset)
  \[
  \{\text{father}(x, y), \text{father}(x, z), \text{father}(z, y), \text{mother}(x, y), \text{mother}(x, z), \text{mother}(z, y)\}
  \]
  - $\text{father}(x, y) \rightarrow \text{grandfather}(x, y)$ covers 0 positives (and 5 negatives)
  - $\text{father}(x, z) \rightarrow \text{grandfather}(x, y)$ covers 5 positives (and 22 negatives)
  - $\text{father}(z, y) \rightarrow \text{grandfather}(x, y)$ covers 2 positive (and 43 negatives)
  - $\text{mother}(x, y) \rightarrow \text{grandfather}(x, y)$ covers 0 positives (and 3 negatives)
  - $\text{mother}(x, z) \rightarrow \text{grandfather}(x, y)$ covers 0 positives (and 18 negatives)
  - $\text{mother}(z, y) \rightarrow \text{grandfather}(x, y)$ covers 3 positives (24 negatives)
FOIL - example

- We continue with the rule $father(x, z) \rightarrow grandfather(x, y)$
- We have several candidate clauses (we only evaluate a subset)
  \[
  \{father(x, y), father(z, y), father(y, z), mother(x, y), mother(z, y), mother(y, z)\}
  \]
  - $father(x, z) \land father(x, y) \rightarrow grandfather(x, y)$ covers 0 positives (and 4 negatives)
  - $father(x, z) \land father(z, y) \rightarrow grandfather(x, y)$ covers 2 positives (and 0 negatives)
  - $father(x, z) \land father(y, z) \rightarrow grandfather(x, y)$ covers 0 positive (and 0 negatives)
  - $father(x, z) \land mother(x, y) \rightarrow grandfather(x, y)$ covers 0 positives (and 0 negatives)
  - $father(x, z) \land mother(z, y) \rightarrow grandfather(x, y)$ covers 3 positives (and 0 negatives)
  - $father(x, z) \land mother(y, z) \rightarrow grandfather(x, y)$ covers 0 positives (and 0 negatives)
The best rule is $father(x, z) \land mother(z, y) \rightarrow grandfather(x, y)$

If we delete the positive examples and repeat the process we will obtain the other rule:

$$father(x, z) \land father(z, y) \rightarrow grandfather(x, y)$$