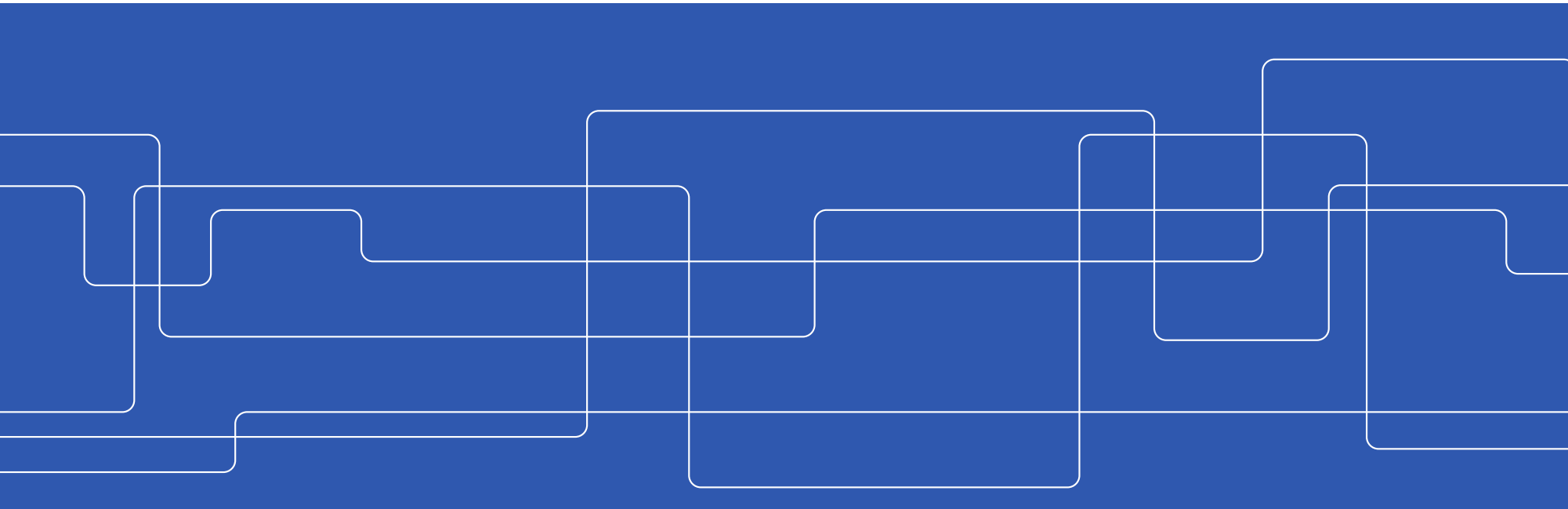
ID2214/FID3214 2020

Programming for Data Science – Decision Trees and Rules

Henrik Boström

Prof. of Computer Science - Data Science Systems
Division of Software and Computer Systems
Department of Computer Science
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
bostromh@kth.se



Outline

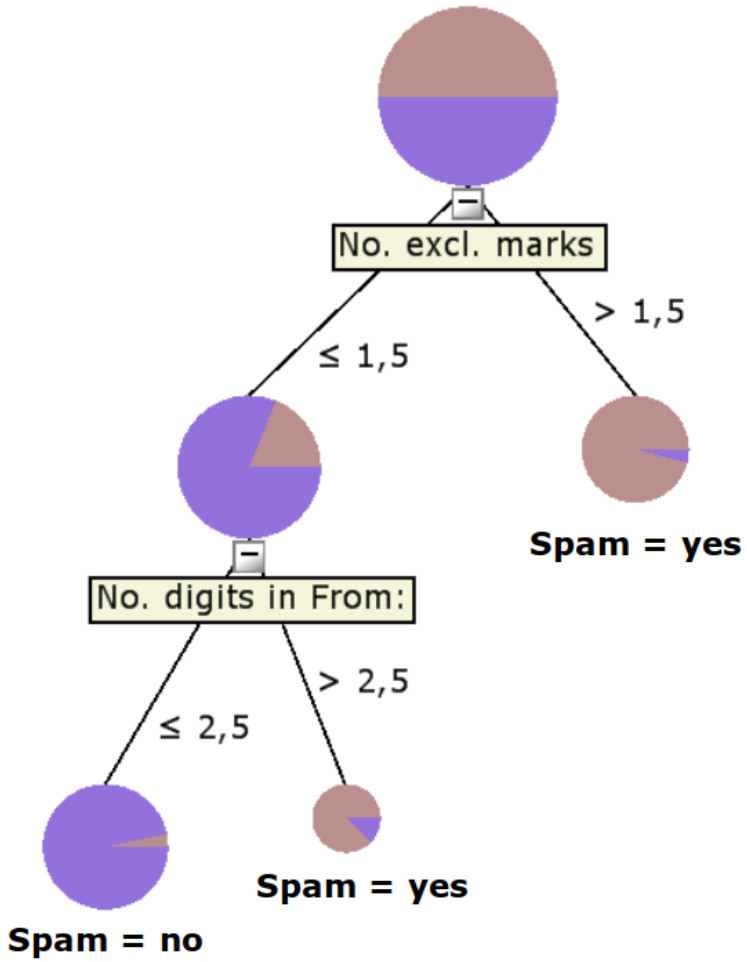
Decision tree learning

- Divide-and-conquer
- Evaluating splits
- Handling numeric features
- Handling missing values
- Overfitting
- The replication problem

Rule learning

- Covering
- Incremental reduced error pruning

Decision tree



Some prominent decision tree learning systems

CART L. Breiman, J. Friedman, R. Olshen and C. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

ID3 J.R. Quinlan. 1986. Induction of decision trees. *Machine learning*, 1(1), pp.81-106.

C4.5 J.R. Quinlan. 1993. *C4.5: Programs for machine learning*. Morgan Kaufmann

J48 Implementation of C4.5 in WEKA

The Divide-and-Conquer algorithm[1]

```
Input: instances I = {(X_1,y_1), ..., (X_n,y_n)},
       features F
Output: a decision tree T

1. if Terminate(I,F) then return T = LeafNode(I)
2. c_1, ..., c_v = a set of conditions
3. I_1, ..., I_v = a partitioning (split) of I such that
                   c_i holds for each instance in I_i
4. for I_i in I_1, ..., I_v:
      T_i = DivideAndConquer(I_i,F)
   let T be a tree with a root from which there is
   an edge, labeled with c_i, to each sub-tree T_i
   return T
```

[1] Also known as *recursive partitioning*

The termination condition

The termination condition checks whether the instances at a current node should be further split or not, and typically evaluates to true if any of the following holds:

- all instances share the same label[1]
- the number of instances are less than a specified threshold, e.g., `min_samples_split`
- the node is at a certain depth in the tree, e.g., `max_depth`
- there is no way of splitting the instances, so that each partition contains at least a specified number of instances, e.g., `min_samples_leaf`

[1] the node is said to be *pure*

The leaf nodes

A leaf node is inserted in the decision tree, whenever the termination condition is satisfied. The leaf node is created using the local instances and may be labeled with:

- in the case of classification, either the most frequent class label among the local instances[2] or a class probability distribution formed using the local class label counts[3]
- in the case of regression, the mean of the target values of the local instances[4]
- in the case that the set of local instances is empty, a label formed from the information in the parent node

[2] resulting in classification trees
[3] resulting in probability estimation trees (PETs)
[4] resulting in regression trees

Finding a set of conditions

A set of conditions is selected, by which the instances are split, i.e., each instance falls into the partition for which it meets the corresponding condition. This set of conditions normally have the following properties:

- the conditions are exclusive and exhaustive, i.e., each instance meets exactly one of the conditions[5]
- all conditions in the set involve a single feature (f)[6]

[5] except if the feature value is missing for the instance

[6] except in so-called *oblique decision trees*

Finding a set of conditions (cont.)

- if the feature f is categorical, the conditions either correspond to a binary test, for some categorical value v:

$$\begin{cases} f = v \\ f \neq v \end{cases}$$

or to a multi-way test, with one condition for each possible categorical value $v_1, \ldots, v_l$ of the feature:

$$\begin{cases} f = v_1 \\ \vdots \\ f = v_l \end{cases}$$

Finding a set of conditions (cont.)

- if the feature f is numerical, the conditions normally correspond to a binary test with respect to some threshold value v:

$$\begin{cases} f \leq v \\ f > v \end{cases}$$

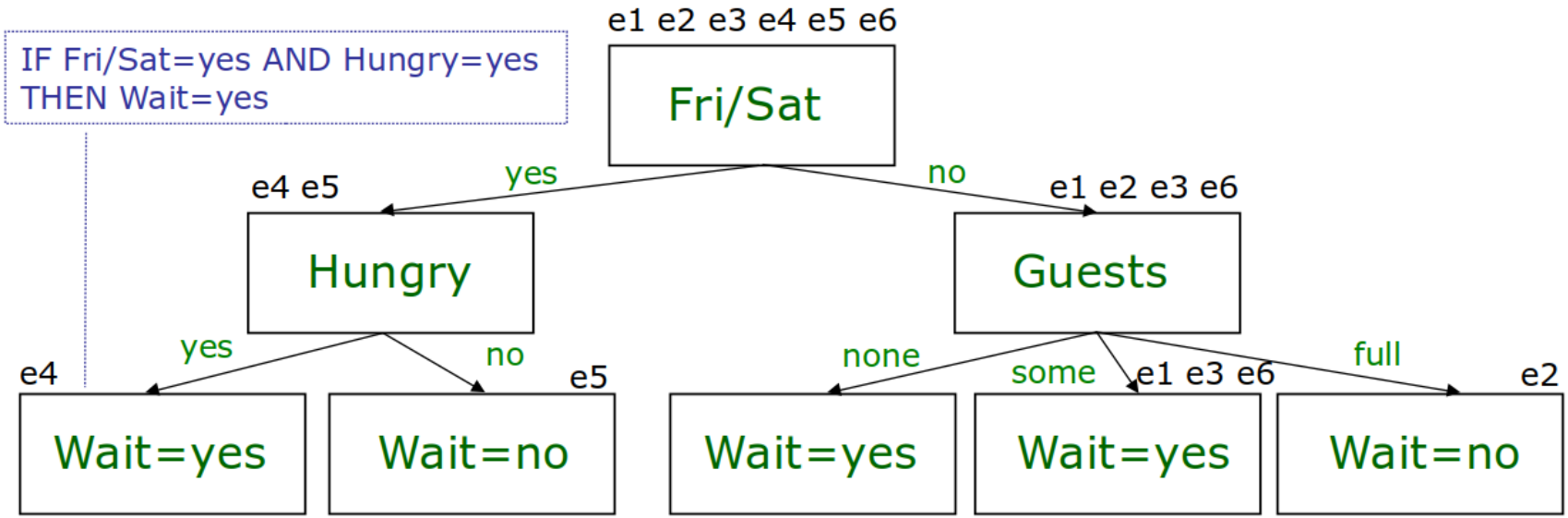
The restaurant example (again)

Ex.	Other	Bar	Fri/Sat	Hungry	Guests	Wait
e1	yes	no	no	yes	some	yes
e2	yes	no	no	yes	full	no
e3	no	yes	no	no	some	yes
e4	yes	no	yes	yes	full	yes
e5	yes	no	yes	no	none	no
e6	no	yes	no	yes	some	yes

Let us consider divide-and-conquer with:

- multi-way splits
- termination on pure or empty nodes
- class labels in the leaf nodes

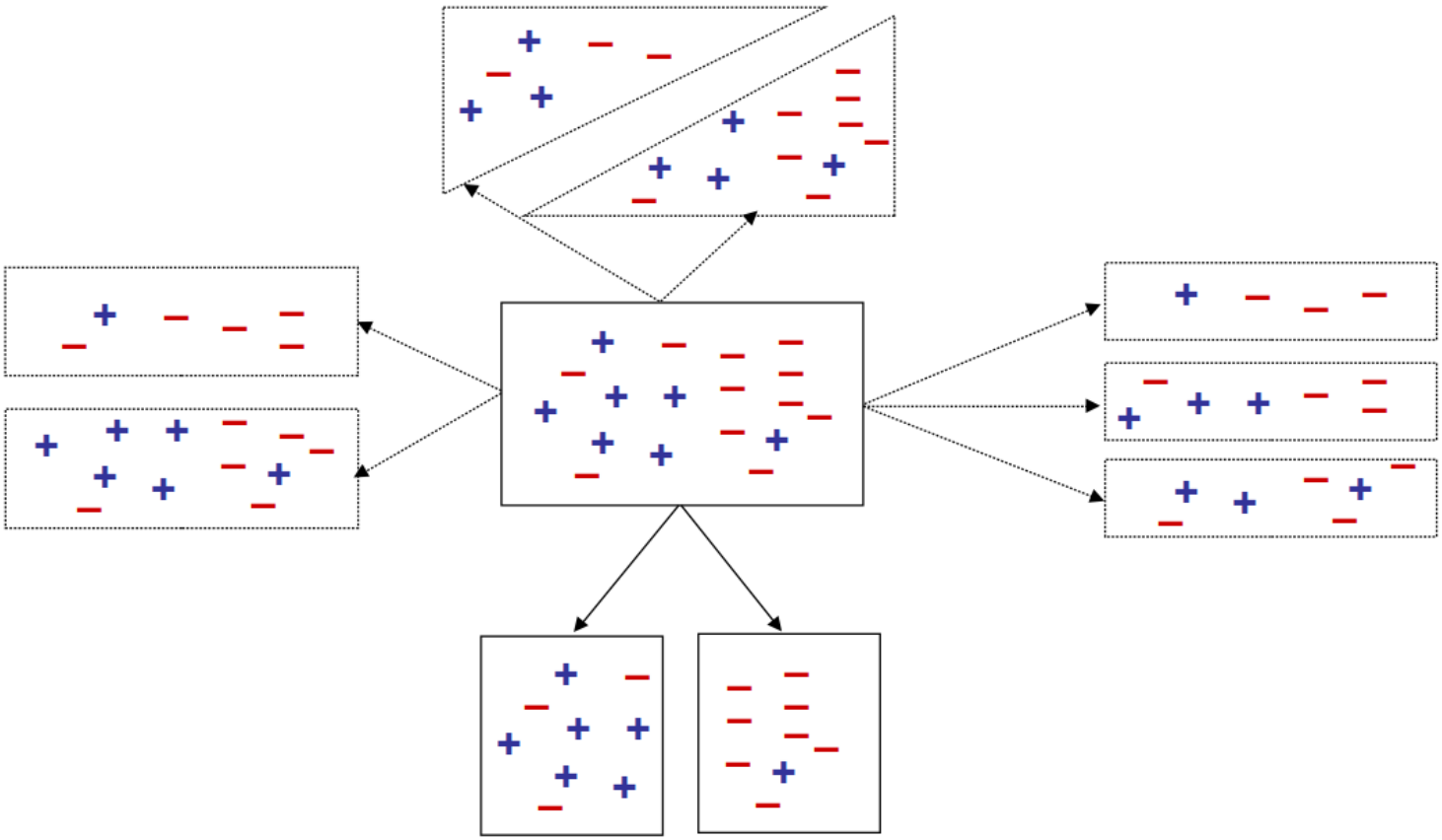
Decision tree (example)



Interpretability

- Can we interpret the model/understand the predictions?
  - To be discussed ...

Evaluating splits



Choosing the split that minimizes impurity

- The impurity of a split s of instances I, resulting in v partitions, can be defined as:

$$Impurity(s) = \sum_{i=1}^{v} \frac{|I_i|}{|I|} Imp(I_i)$$

- For regression problems (numerical labels), the impurity of a set of instances $\{(X_1, y_1), \ldots, (X_n, y_n)\}$, is often defined by:

$$Imp(\{(X_1, y_1), \ldots, (X_n, y_n)\}) = \frac{\sum_{i=1}^{n} (\bar{y} - y_i)^2}{n}$$

- For classification problems (categorical labels), two common choices are *information gain (entropy)* and *the Gini index*.

Measuring impurity by entropy

- Assume the possible events $e_1, \ldots, e_k$ have the probabilities $P(e_1), \ldots, P(e_k)$
- The optimal number of bits that are required to encode a particular event e_i is $-log_2(P(e_i))$
- The average number of bits to encode what event occurs, i.e., the *entropy*, is:

$$E(P(e_1), \ldots, P(e_k)) = -\sum_{i=1}^{k} P(e_i) log_2(P(e_i))$$

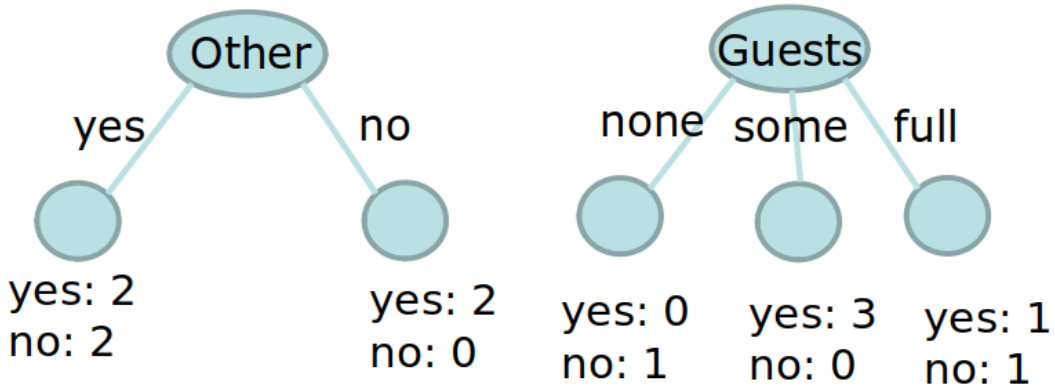
- Given a set of instances $I = \{(X_1, y_1), \ldots, (X_n, y_n)\}$ and class labels $c_1, \ldots, c_k$:

$$Imp(\{(X_1, y_1), \ldots, (X_n, y_n)\}) = E(P(c_1), \ldots, P(c_k))$$

where $P(c_i) = |\{y_j \in \{y_1, \ldots, y_n\} : y_j = c_i\}|/n$

Choosing the most informative split (example)

Other	Guests	Wait
yes	some	yes
yes	full	no
no	some	yes
yes	full	yes
yes	none	no
no	some	yes



$$Impurity(Other) = \frac{4}{6}E(\frac{2}{4},\frac{2}{4}) + \frac{2}{6}E(\frac{2}{2},\frac{0}{2}) = \frac{4}{6}$$

$$Impurity(Guests) = \frac{1}{6}E(\frac{0}{1},\frac{1}{1}) + \frac{3}{6}E(\frac{3}{3},\frac{0}{3}) + \frac{2}{6}E(\frac{1}{2},\frac{1}{2}) = \frac{2}{6}$$

Measuring impurity by Gini index

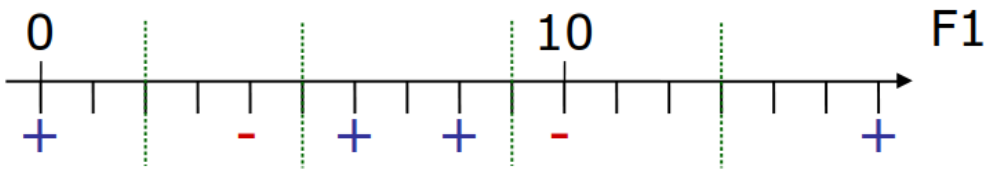
Given a set of instances $I = \{(X_1, y_1), \ldots, (X_n, y_n)\}$ and class labels $c_1, \ldots, c_k$:

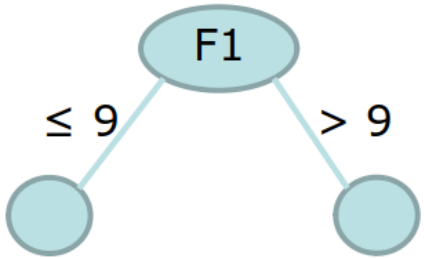
$$Imp(\{(X_1, y_1), \ldots, (X_n, y_n)\}) = 1 - \sum_{i=1}^{k} P(c_i)^2$$

where $P(c_i) = |\{y_j \in \{y_1, \ldots, y_n\} : y_j = c_i\}|/n$

Handling numeric features

Ex	F1	...	Class
e1	0		+
e2	4		-
e3	6		+
e4	8		+
e5	10		-
e6	16		+





- Discretization before tree generation
- Discretization during tree generation

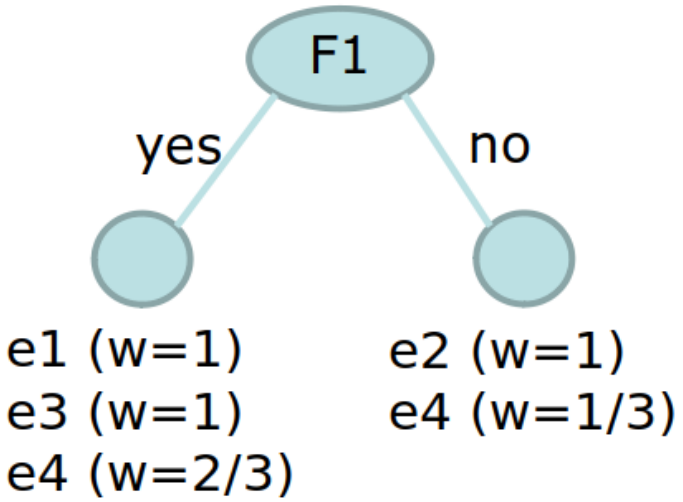
Handling missing values

Different approaches to handling missing values in decision tree learning:

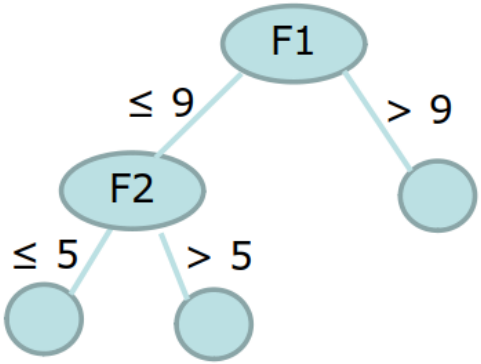
- Remove features with missing values
- Remove instances with missing values
- Impute values, e.g., assume most frequent value
- Distribute instances over multiple nodes

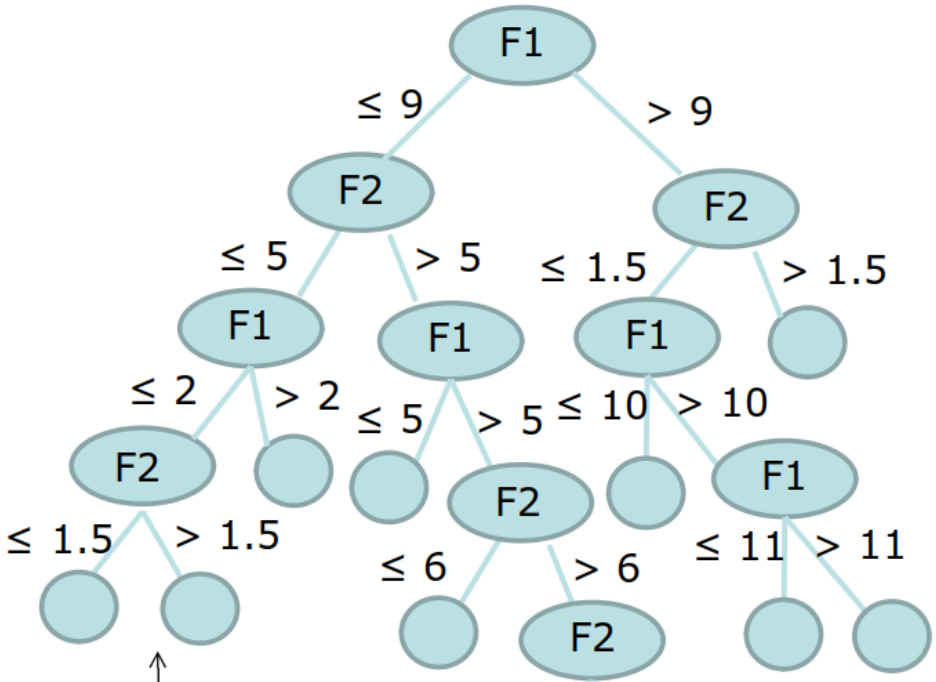
Distributing instances over multiple nodes

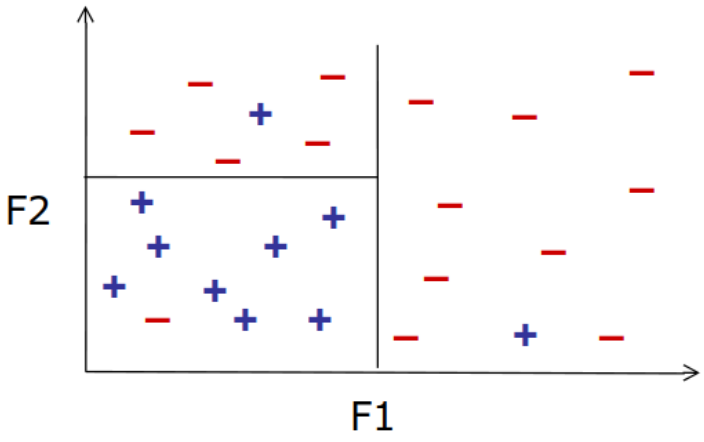
Example	F1	...	Class
e1	yes		+
e2	no		+
e3	yes		-
e4	?		-

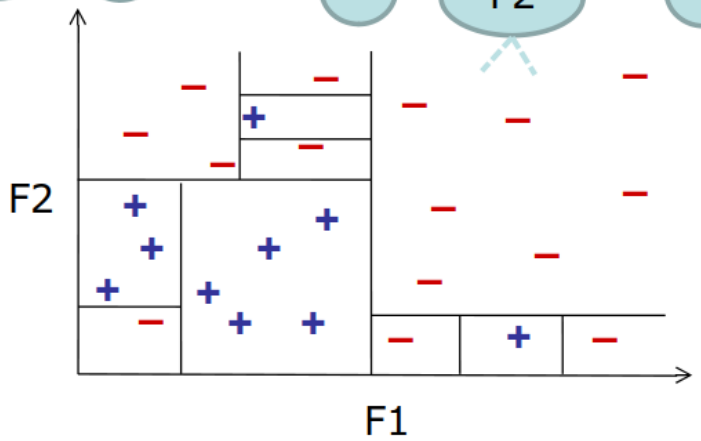


Overfitting



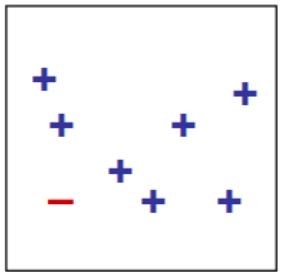




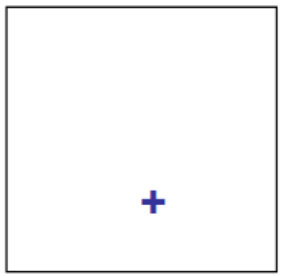


Estimating class probabilities

$$P_{rel.freq.}(+) = \frac{|E_+|}{|E|} \qquad P_{Laplace}(+) = \frac{|E_+|+1}{|E|+k} \qquad P_m(+) = \frac{|E_+|+f_+ \cdot m}{|E|+m}$$

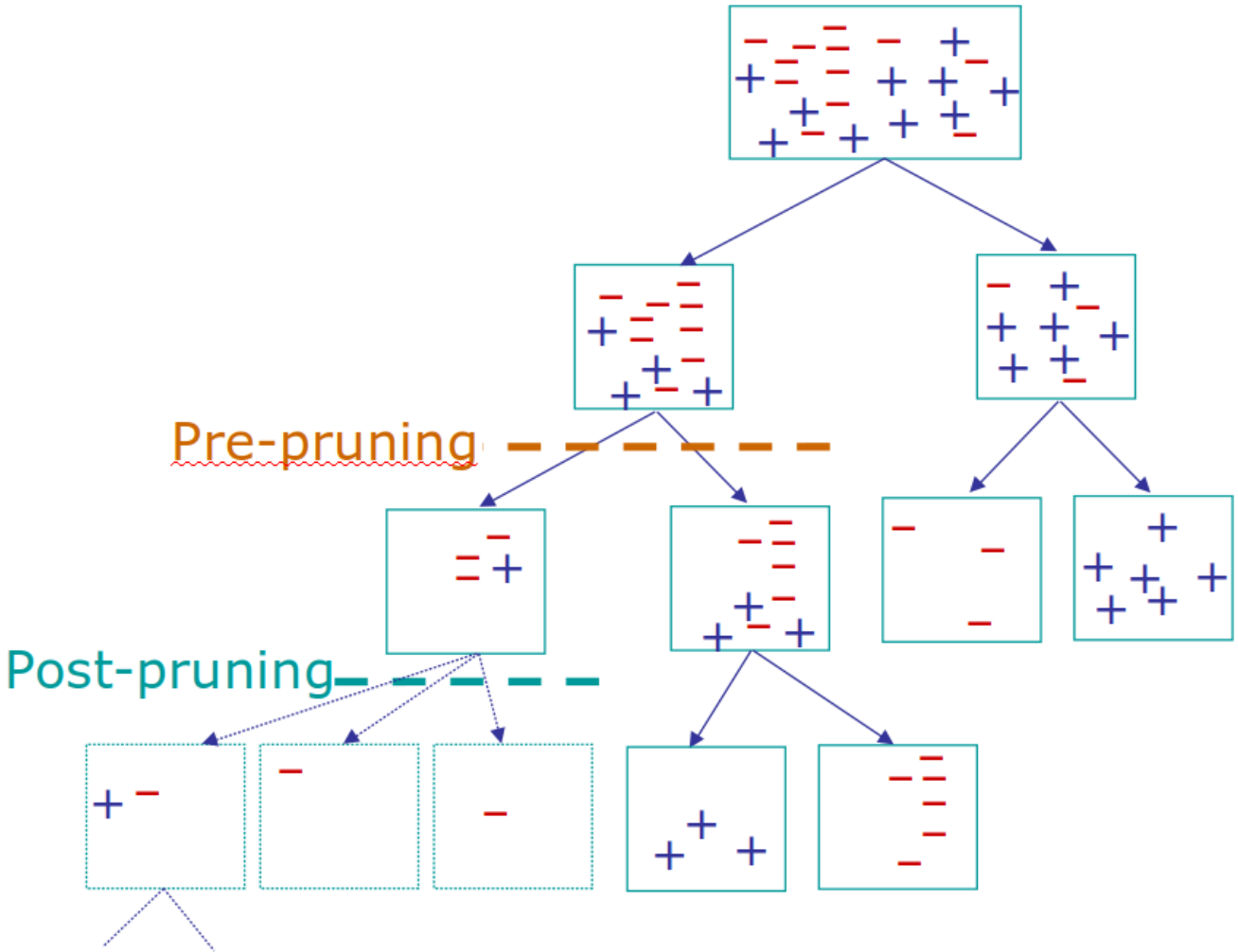


$$P_{rel.freq.}(+) = \frac{7}{8} \qquad P_{Laplace}(+) = \frac{7+1}{8+2} \qquad P_m(+) = \frac{7+f_+ \cdot m}{8+m}$$



$$P_{rel.freq.}(+) = \frac{1}{1} \qquad P_{Laplace}(+) = \frac{1+1}{1+2} \qquad P_m(+) = \frac{1+f_+ \cdot m}{1+m}$$

Pre-pruning vs. post-pruning



Decision trees in Scikit-learn

```
import numpy as np
import pandas as pd
from sklearn import tree
```

```
# Assuming that "glass_train.csv" and "glass_test.csv" are in the current directory
# Download the files from assignment 1 or 2

glass_train_df = pd.read_csv("glass_train.csv")
y = glass_train_df["CLASS"].values

glass_train_df.drop(["CLASS","ID"],axis="columns",inplace=True)
X = glass_train_df.values

dt = tree.DecisionTreeClassifier(max_depth=3)

dt.fit(X,y)
```

Decision trees in Scikit-learn (cont.)

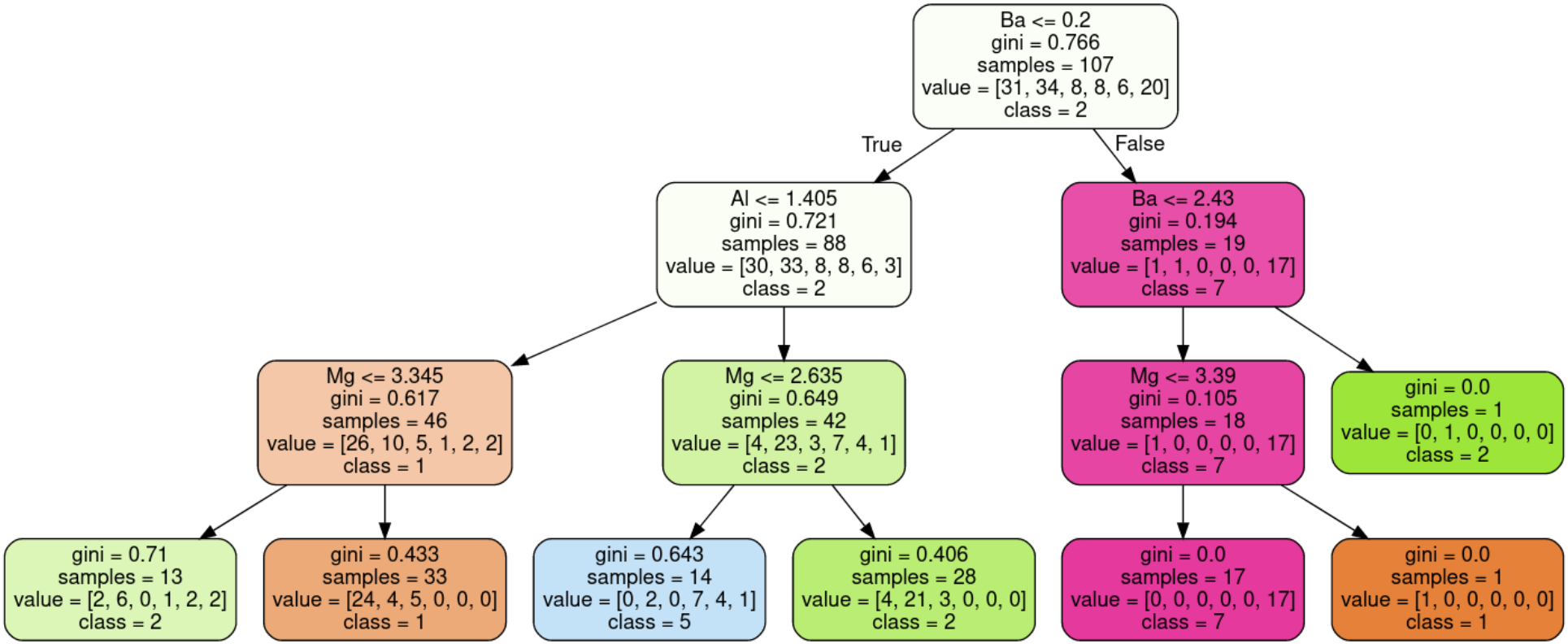
```
tree_desc = tree.export_text(dt, feature_names=list(glass_train_df.columns))
print(tree_desc)
```

```
|--- Ba <= 0.20
|   |--- Al <= 1.40
|   |   |--- Mg <= 3.34
|   |   |   |--- class: 2
|   |   |--- Mg >  3.34
|   |   |   |--- class: 1
|   |--- Al >  1.40
|   |   |--- Mg <= 2.64
|   |   |   |--- class: 5
|   |   |--- Mg >  2.64
|   |   |   |--- class: 2
|--- Ba >  0.20
|   |--- Mg <= 3.39
|   |   |--- Ca <= 11.38
|   |   |   |--- class: 7
|   |   |--- Ca >  11.38
|   |   |   |--- class: 2
|   |--- Mg >  3.39
|   |   |--- class: 1
```

Decision trees in Scikit-learn (cont.)

```
# Install graphviz first, e.g. with "conda install python-graphviz"

import graphviz
dot_data = tree.export_graphviz(dt, feature_names=list(glass_train_df.columns),
                                class_names = [str(c) for c in dt.classes_],
                                filled=True, rounded=True)
graph = graphviz.Source(dot_data)
display(graph)
```

Decision trees in Scikit-learn (cont.)



Decision trees in Scikit-learn (cont.)

```
glass_test_df = pd.read_csv("glass_test.csv")
test_y = glass_test_df["CLASS"].values
test_X = glass_test_df.drop(["CLASS","ID"],axis="columns").values

predictions = dt.predict(test_X)
display(predictions)

print("Accuracy: {:.4f}".format(np.sum(test_y==predictions)/len(predictions)))
```

```
array([2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 7, 7, 1, 1, 1, 1, 1, 2, 1, 7, 7, 2,
       5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 5, 1, 1, 2, 2, 2, 2,
       7, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 7, 7, 1, 1, 1, 2,
       1, 7, 5, 5, 2, 2, 2, 2, 2, 2, 1, 1, 7, 2, 5, 1, 1, 1, 1, 1, 2, 2,
       2, 7, 1, 1, 5, 1, 1, 1, 1, 1, 1, 7, 1, 1, 2, 2, 1, 2, 2])
Accuracy: 0.6636
```

Decision trees in Scikit-learn (cont.)

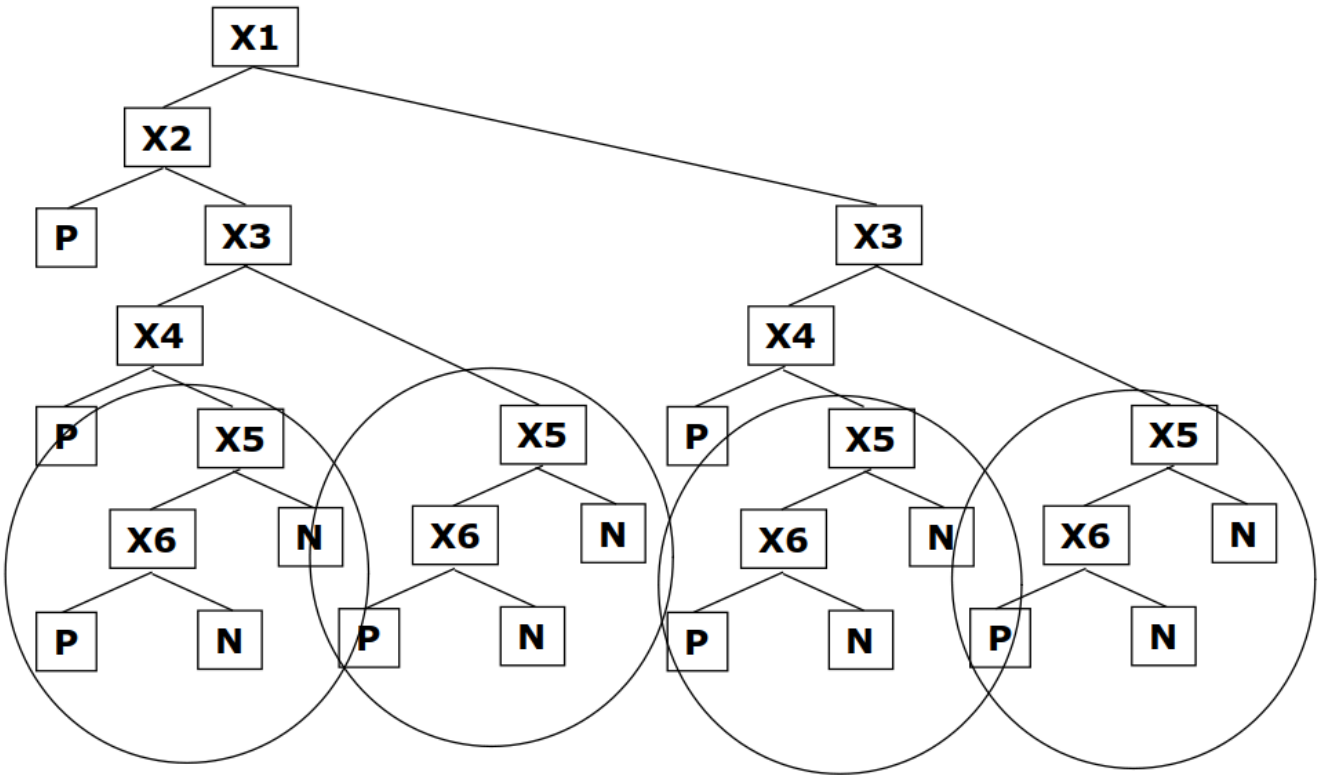
```
predictions = dt.predict_proba(test_X)

df = pd.DataFrame(predictions,columns=dt.classes_)
display(df)
```

	1	2	3	5	6	7
0	0.142857	0.750000	0.107143	0.000000	0.000000	0.000000
1	0.153846	0.461538	0.000000	0.076923	0.153846	0.153846
2	0.727273	0.121212	0.151515	0.000000	0.000000	0.000000
3	0.727273	0.121212	0.151515	0.000000	0.000000	0.000000
4	0.142857	0.750000	0.107143	0.000000	0.000000	0.000000
...	...	...	...	...	...	...
102	0.142857	0.750000	0.107143	0.000000	0.000000	0.000000
103	0.142857	0.750000	0.107143	0.000000	0.000000	0.000000
104	0.727273	0.121212	0.151515	0.000000	0.000000	0.000000
105	0.153846	0.461538	0.000000	0.076923	0.153846	0.153846
106	0.142857	0.750000	0.107143	0.000000	0.000000	0.000000

107 rows × 6 columns

The replication problem

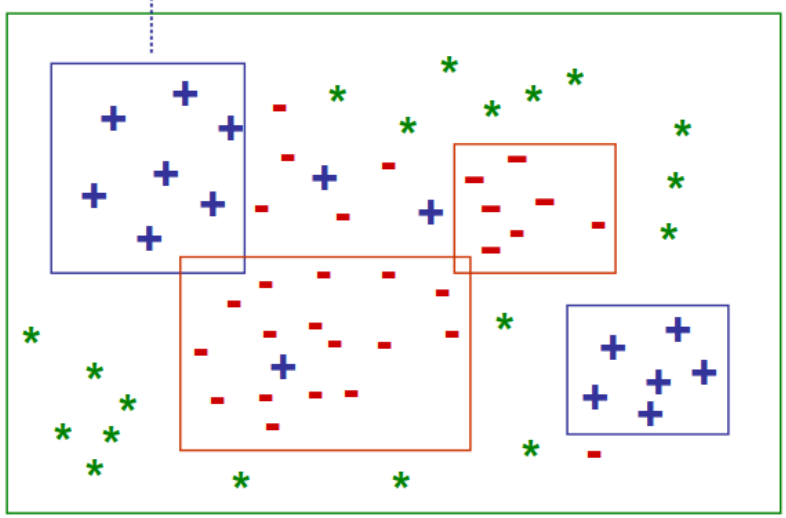
$$X1 = y \,\&\, X2 = y \rightarrow P$$
$$X3 = y \,\&\, X4 = y \rightarrow P$$
$$X5 = y \,\&\, X6 = y \rightarrow P$$
$$otherwise \rightarrow N$$



Rule learning

IF FractionOfRotatableBonds > 0.178
AND LogP ≤ 4.63415
AND PolarSurface ≤ 50.95
THEN Solubility=Good

P(Solubility=Good) = 0.80
P(Solubility=Medium) = 0.10
P(Solubility=Poor) = 0.10



Some prominent rule learning systems

AQ R.S. Michalski. 1969. On the quasi-minimal solution of the covering problem. Proc. of the 5th International Symposium on Information Processing, pp. 125128.

CN2 P. Clark and T. Niblett. 1989. The CN2 induction algorithm. *Machine Learning*, 3, pp. 261-283.

RIPPER W. Cohen. Fast effective rule induction. Proc. of the 12th International Conference on Machine Learning. Morgan Kaufmann, pp 115-123.

JRip Implementation of RIPPER in WEKA

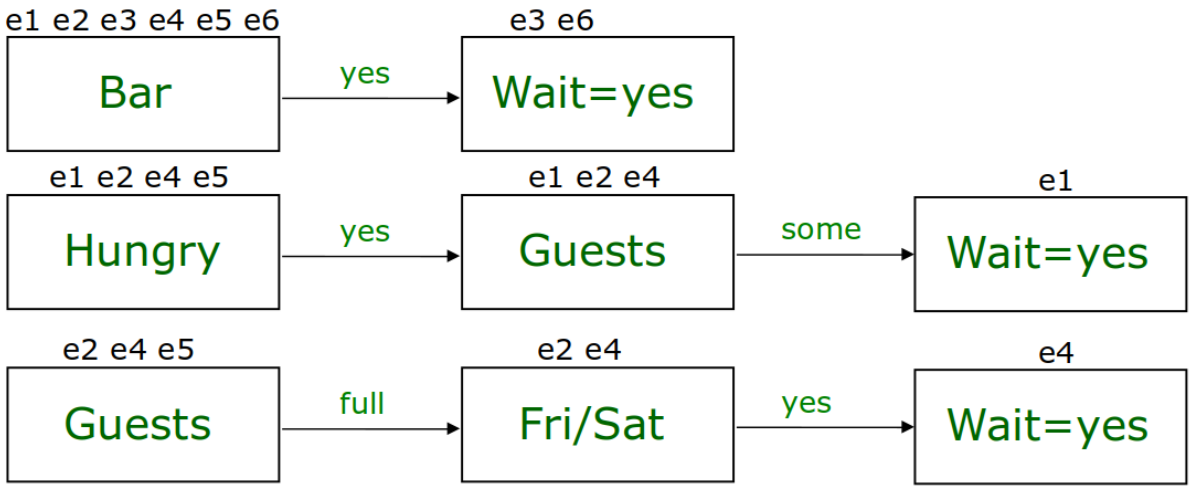
The Covering algorithm[2]

```
Input: instances I, labels c_1, ..., c_k
Output: ordered rule set (decision list) R

for i = 1 to k:
   P = all instances in I labeled with c_i
   while P != {}:
      r := IF THEN c_i
      while r covers some instance in I \ P:
         add condition to r
      add r to R
      I = I \ cover-set(r,I)
      P = all instances in I labeled with c_i
return R
```

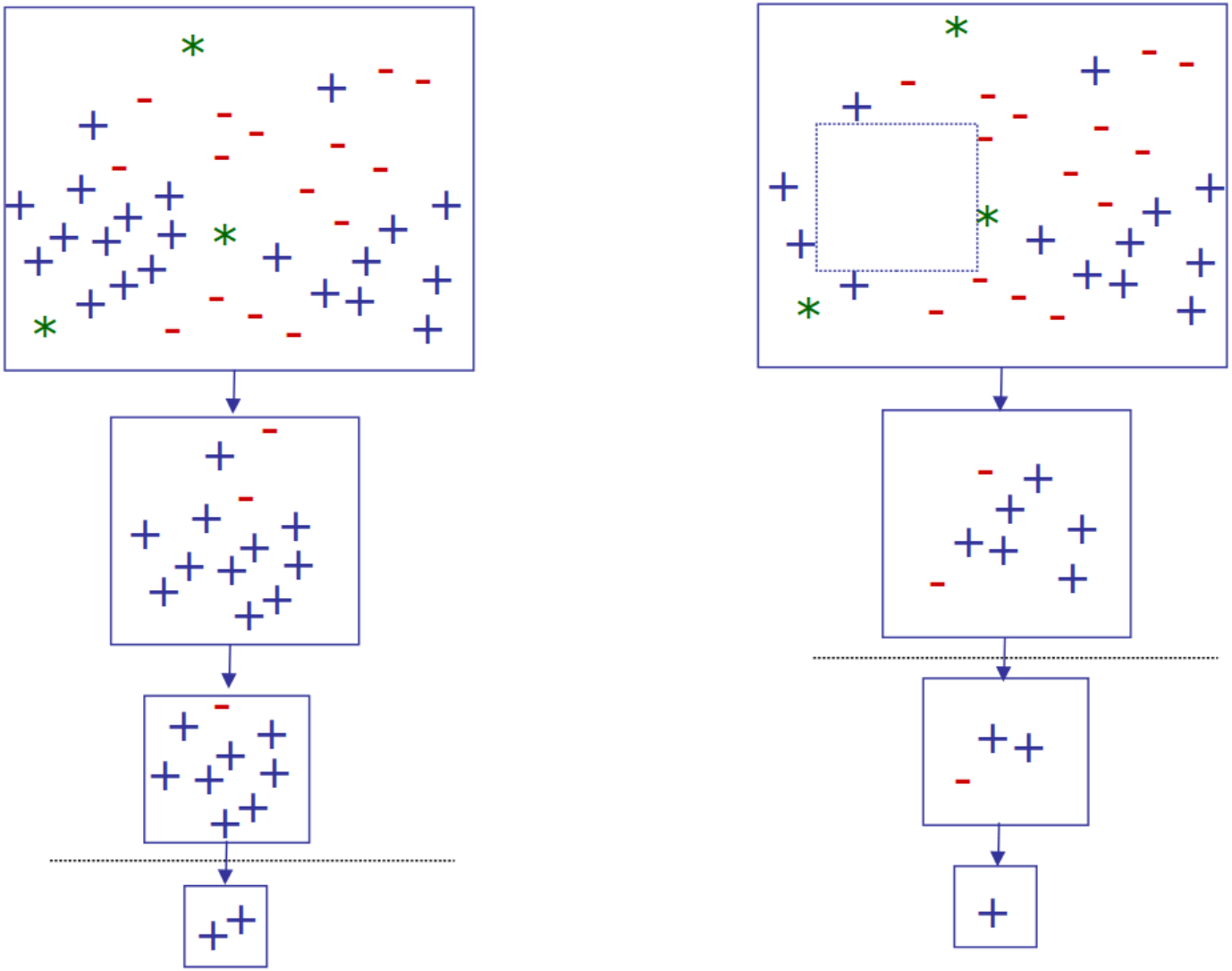
[2] Also known as *separate-and-conquer*

Covering (example)

Ex.	Other	Bar	Fri/Sat	Hungry	Guests	Wait
e1	yes	no	no	yes	some	yes
e2	yes	no	no	yes	full	no
e3	no	yes	no	no	some	yes
e4	yes	no	yes	yes	full	yes
e5	yes	no	yes	no	none	no
e6	no	yes	no	yes	some	yes



Incremental reduced error pruning



Issues with rule learning

- Rules in a decision list cannot be interpreted independently
- For a rule set, multiple (contradictory) rules may apply
  - An ordering may be assumed, e.g. use best rule first
  - Classifications may be combined, however leading to difficulties in interpretation (again)
- For a rule set, none of the rules may apply
  - Class probability estimates from non-covered instances may be used
  - *A priori* probabilities may be used

Summary

- Algorithms for learning decision trees and rules have been considered; divide-and-conquer and covering. The former is more efficient, but may lead to more complex models and lower predictive performance, in particular for disjunctive concepts
- The generated models are interpretable in principle, but in practice, the complexity of the models need to be controlled, perhaps leading to sub-optimal performance[7]
- The algorithms are not the most powerful wrt. predictive performance and quite unstable; small changes in input may lead to large variations in output.

[7] this is often referred to as the *accuracy vs. interpretability trade-off*