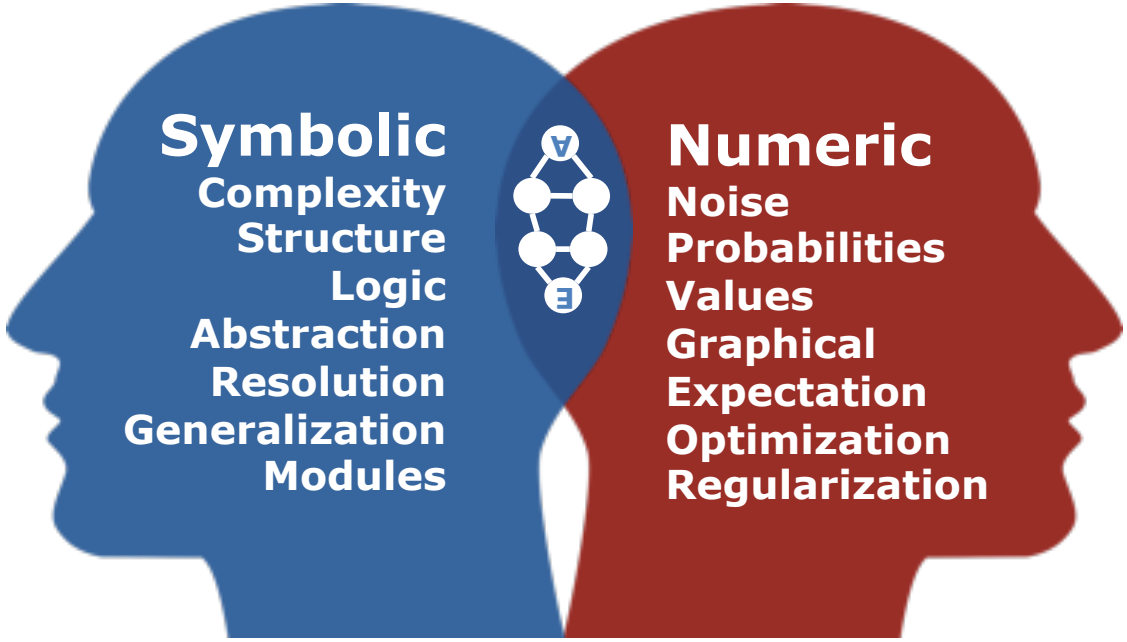
TECHNISCHE UNIVERSITÄT DARMSTADT



Statistical Relational AI

Rule Learning and Inductive Logic Programming

Kristian Kersting

Thanks to Rina Dechter, Luc De Raedt, Pedro Domingos, Peter Flach, Vibhav Gogate, Carlos Guestrin, Daphen Koller, Nir Friedman, Ray Mooney, Sriraam Natarajan, David Poole, Fabrizio Riguzzi, Dan Suciu, Guy van den Broeck, and many others for making their slides publically available

Why learning rules?

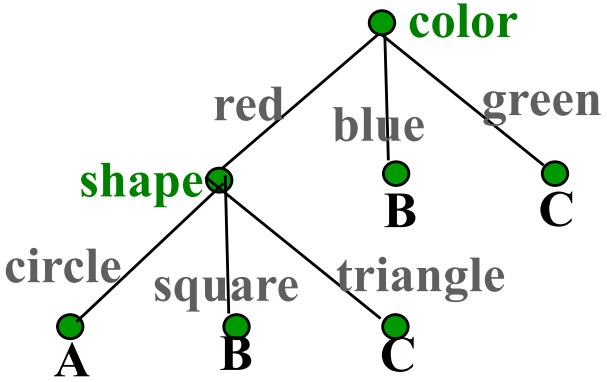
TECHNISCHE UNIVERSITÄT DARMSTADT

- If-then rules in logic are a standard representation of knowledge that have proven useful in expert-systems and other AI systems
- Rules are fairly easy for people to understand and therefore can help provide insight and comprehensible results for human users.
- Methods for automatically inducing rules from data have been shown to build more accurate expert systems than human knowledge engineering for some applications.

Consider e.g. decision trees. They can be turned into rules

TECHNISCHE UNIVERSITÄT DARMSTADT

For each path in a decision tree from the root to a leaf, create a rule with the conjunction of tests along the path as an antecedent and the leaf label as the consequent.



$\text{red} \wedge \text{circle} \rightarrow \text{A}$
$\text{blue} \rightarrow \text{B}$
$\text{red} \wedge \text{square} \rightarrow \text{B}$
$\text{green} \rightarrow \text{C}$
$\text{red} \wedge \text{triangle} \rightarrow \text{C}$

Post-Processing Decision-Tree Rules

TECHNISCHE UNIVERSITÄT DARMSTADT

However, resulting rules may lead to competing conflicting conclusions on some instances.

Sort rules by training (validation) accuracy to create an ordered decision list. The first rule in the list that applies is used to classify a test instance. Use "!" cuts in Prolog

red ∧ circle → A (97% train accuracy)
red ∧ big → B (95% train accuracy)
:
:
Test case: <big, red, circle> assigned to class A

How do we learn rules? A common approach is sequential covering

TECHNISCHE UNIVERSITÄT DARMSTADT

A set of rules is learned one at a time, each time finding a single rule that covers a large number of positive instances without covering any negatives, removing the positives that it covers, and learning additional rules to cover the rest.

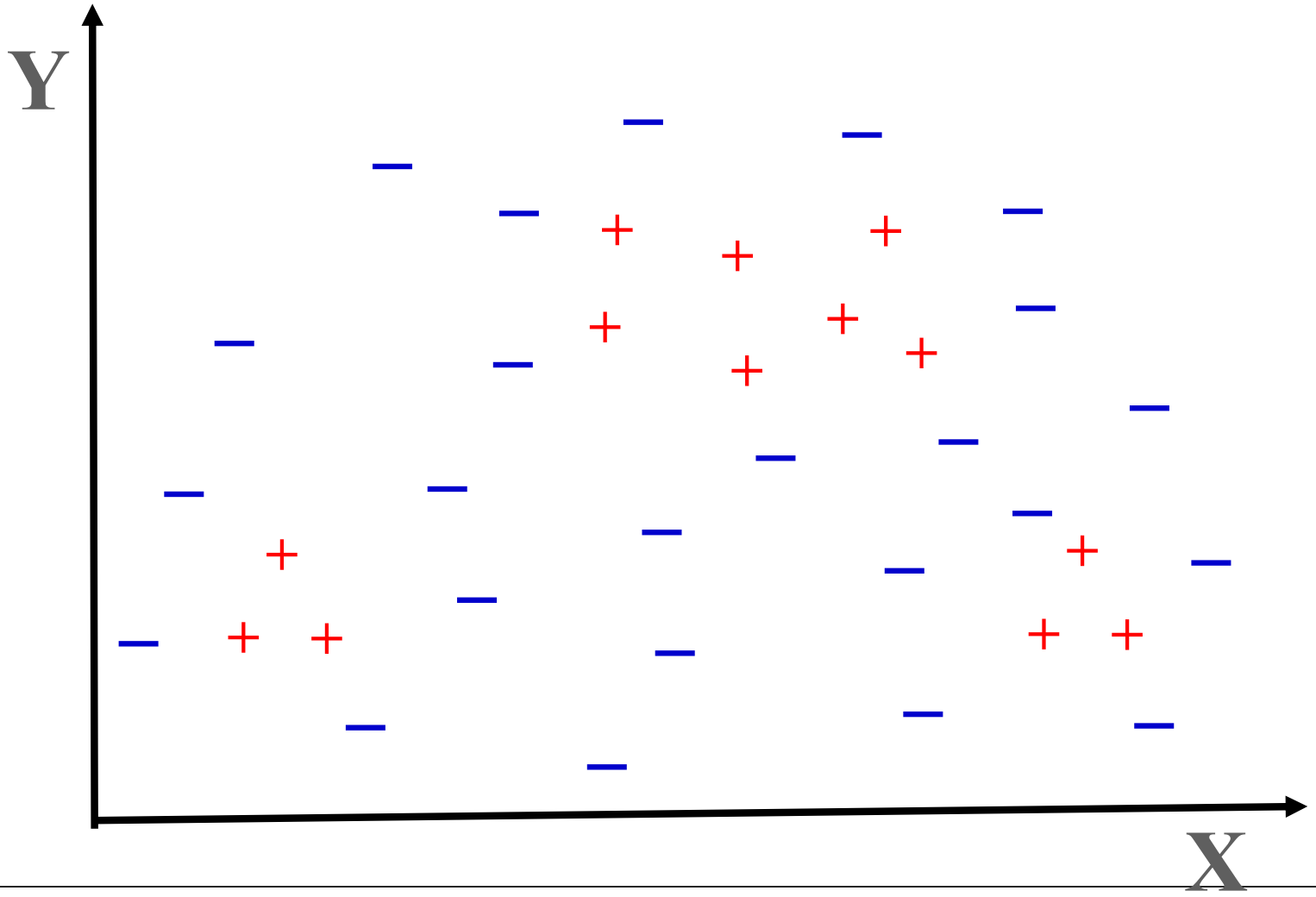
Let *P* be the set of positive examples
  Until *P* is empty do:
    Learn a rule *R* that covers a large number of elements of *P* but no negatives.
    Add *R* to the list of rules.
    Remove positives covered by *R* from *P*

How do we learn rules? A common approach is sequential covering

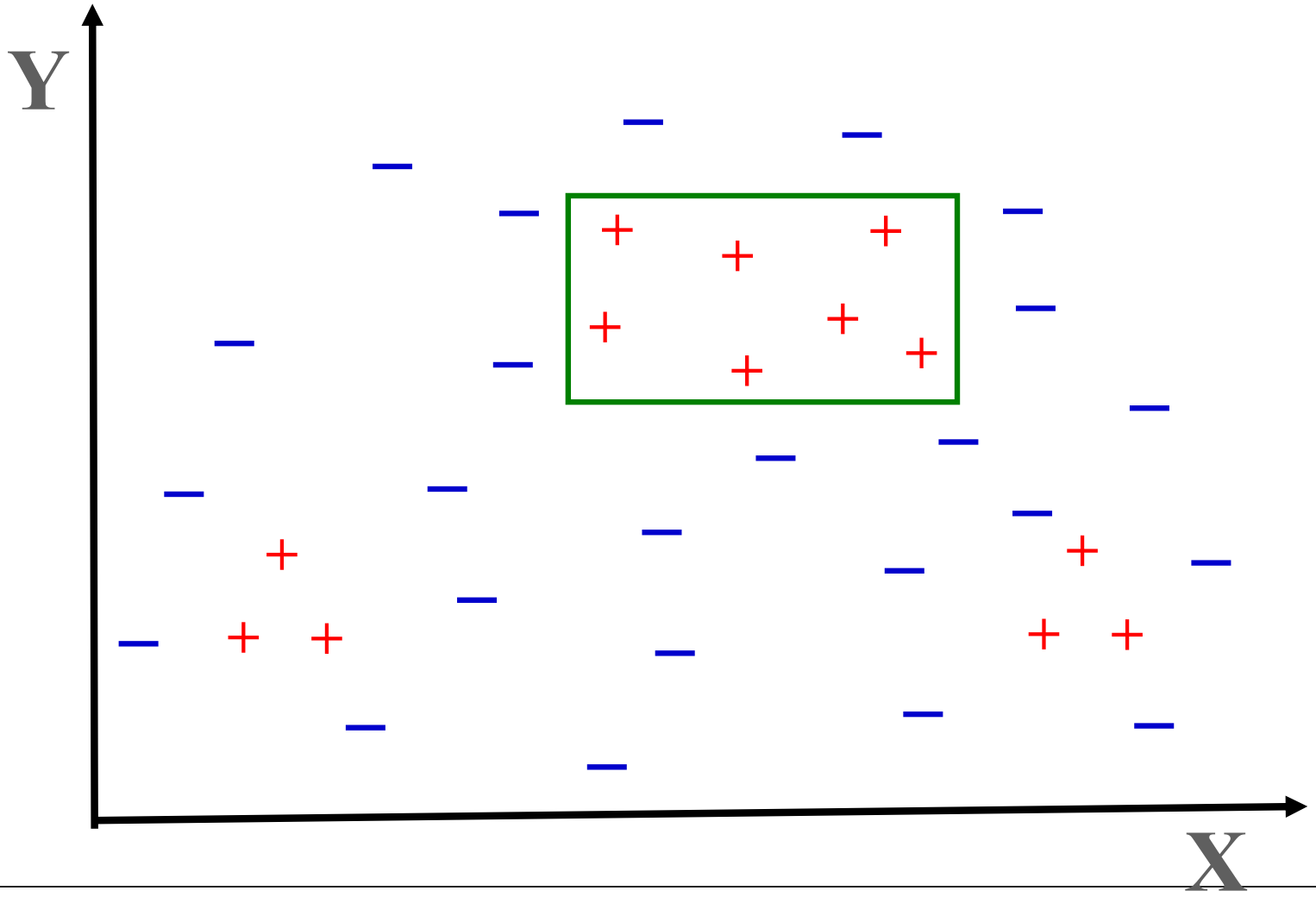
TECHNISCHE UNIVERSITÄT DARMSTADT

- This is an instance of the greedy algorithm for minimum set covering and does not guarantee a minimum number of learned rules.
- Minimum set covering is NP-hard and the greedy algorithm is a standard approximation algorithm.
- Methods for learning individual rules vary.

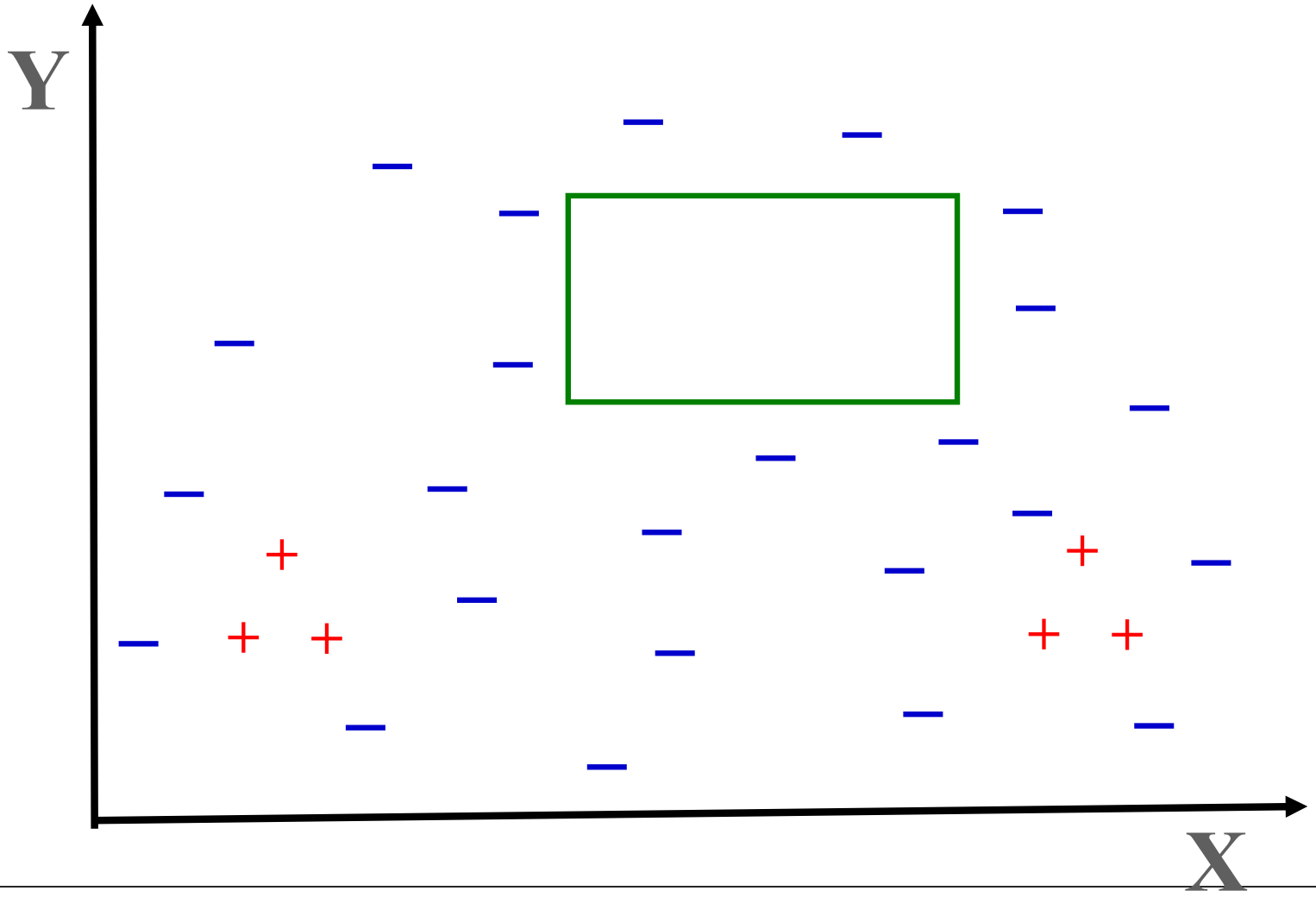
Greedy Sequential Covering Example



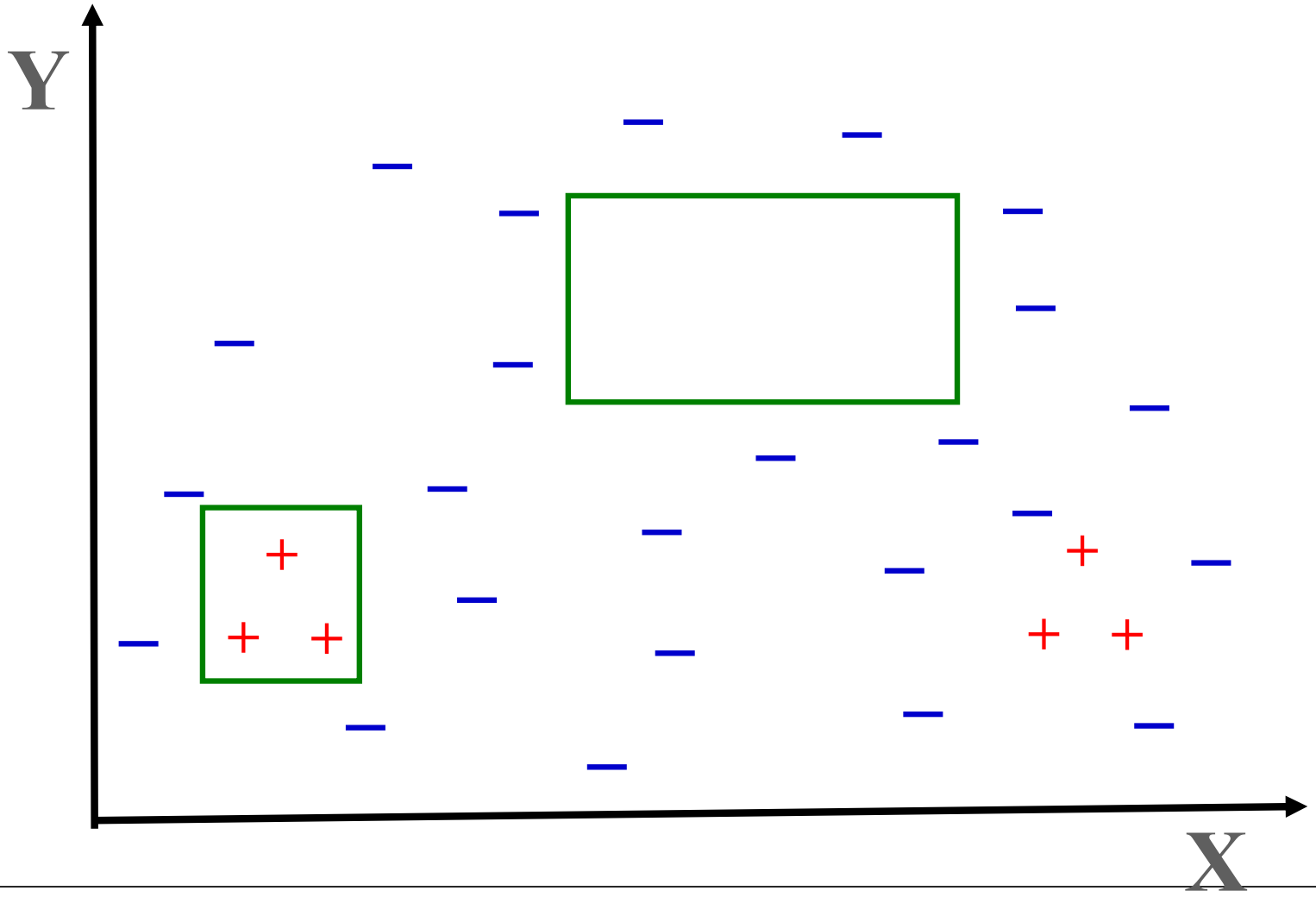
Greedy Sequential Covering Example



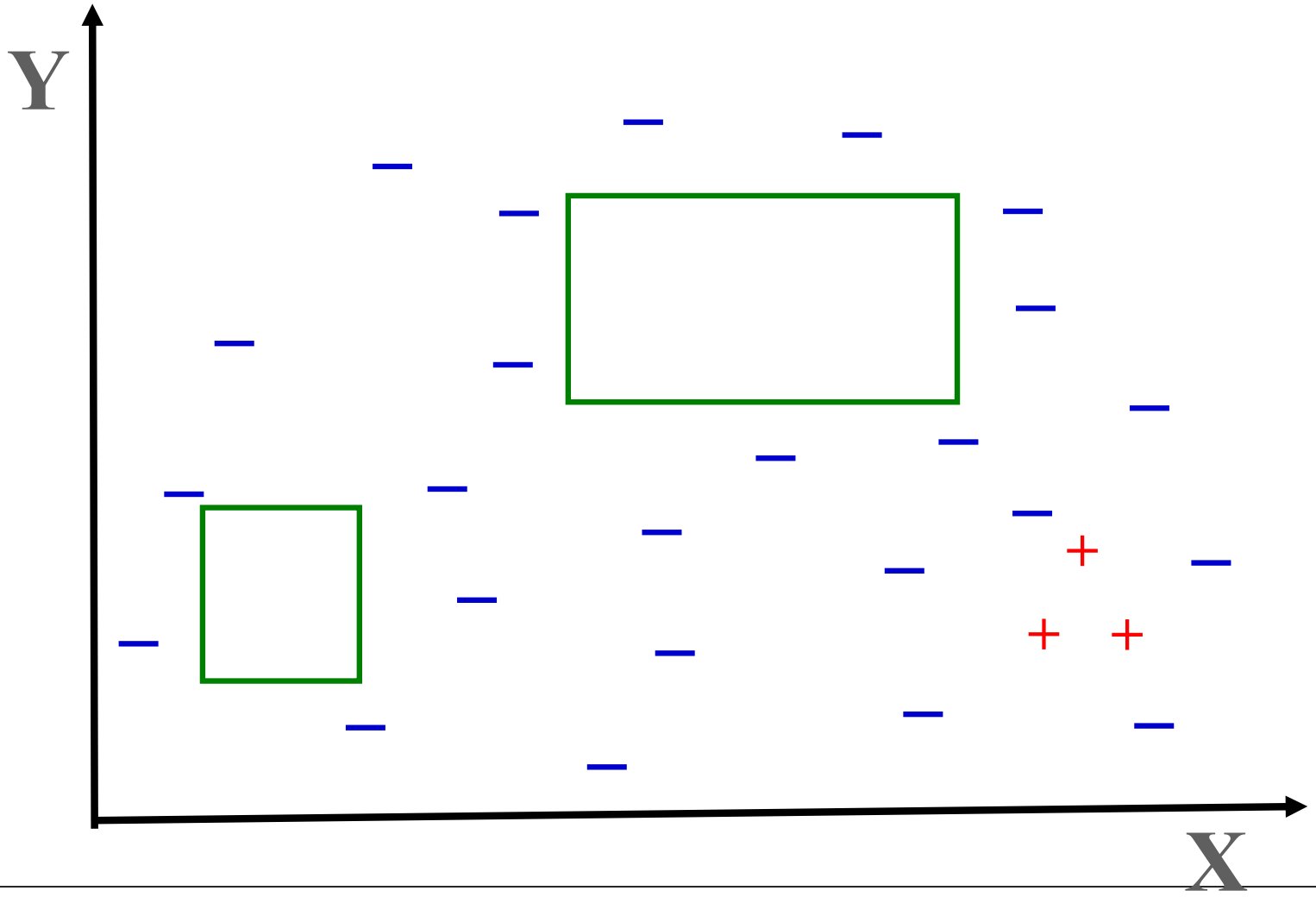
Greedy Sequential Covering Example



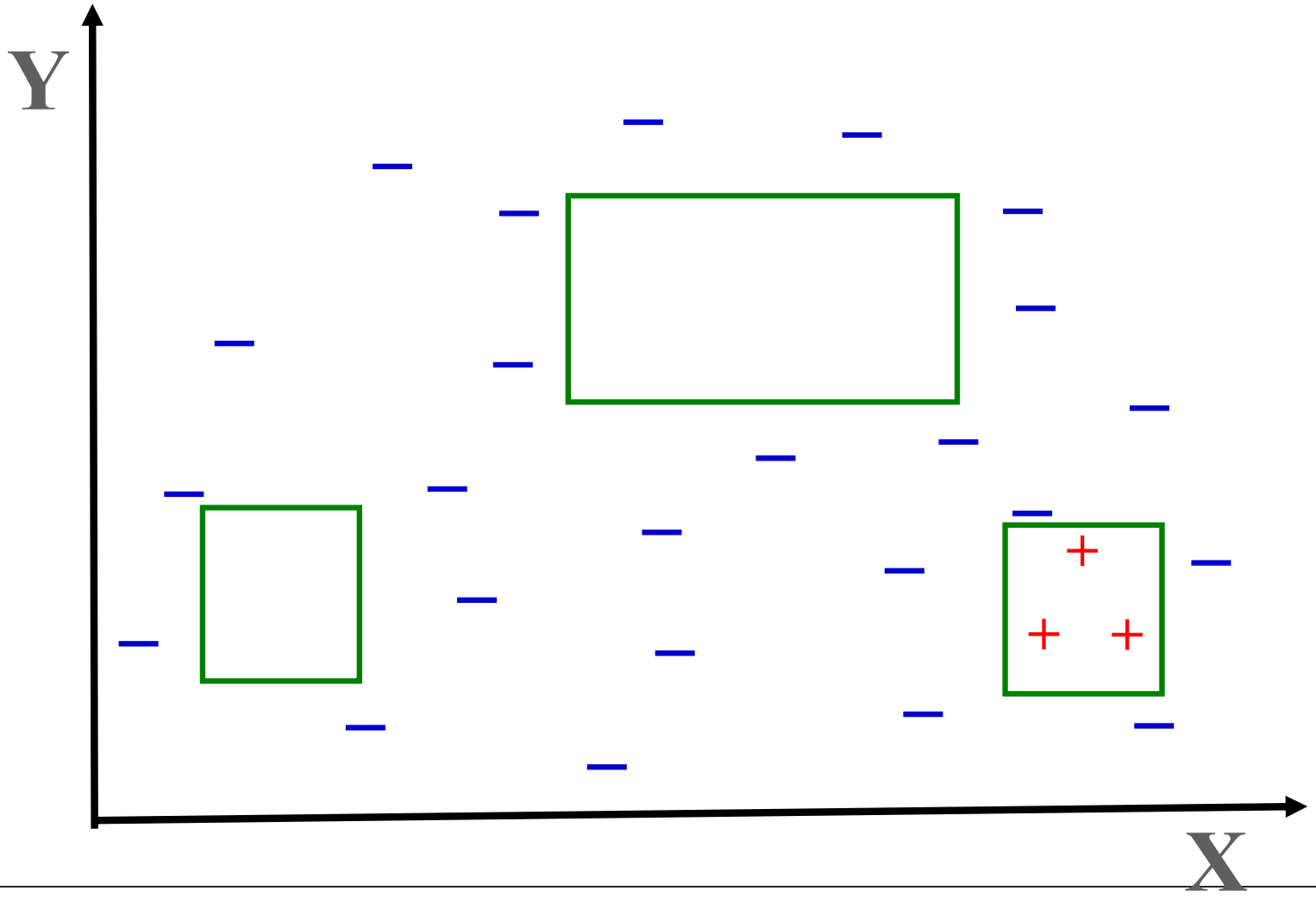
Greedy Sequential Covering Example



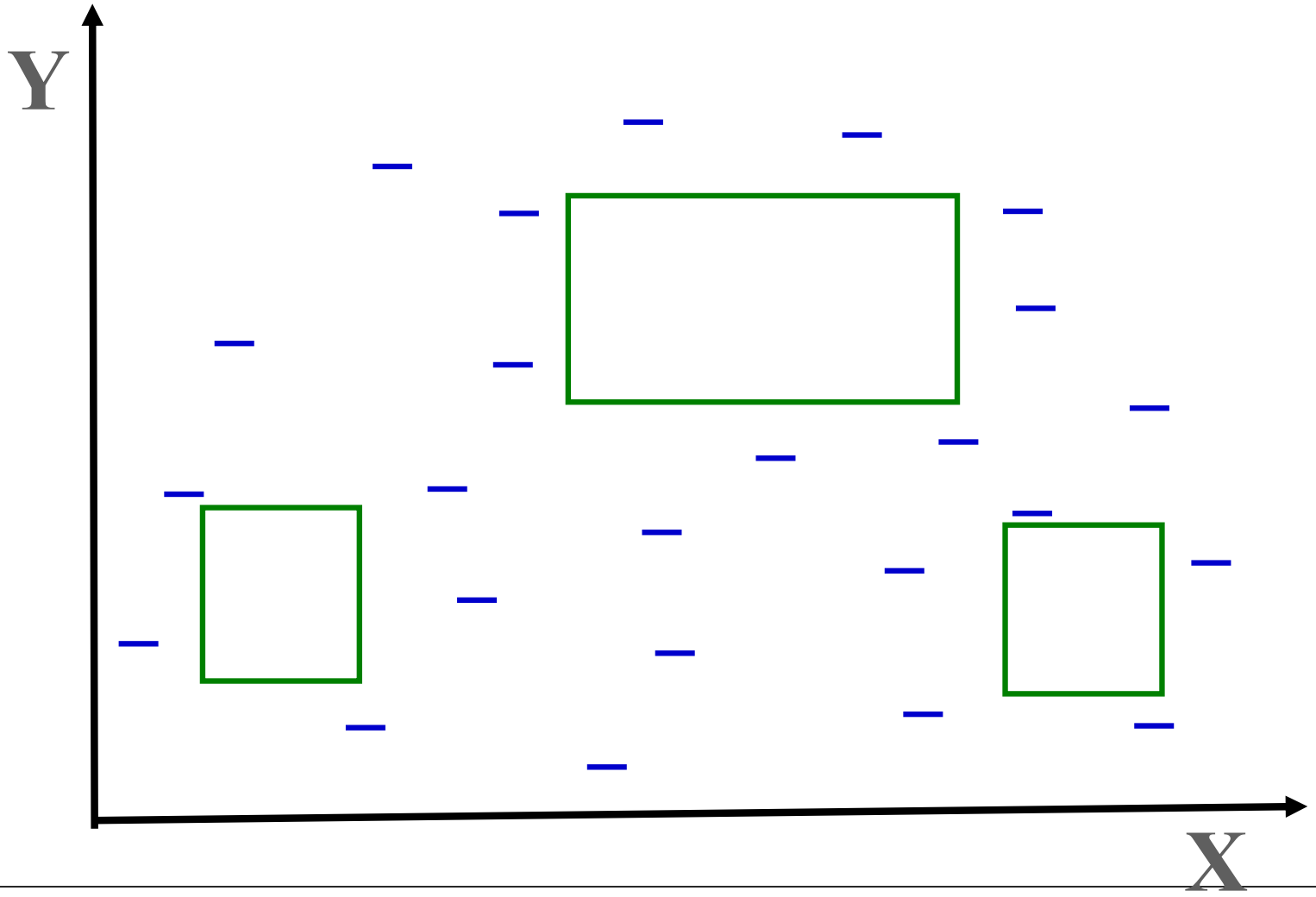
Greedy Sequential Covering Example



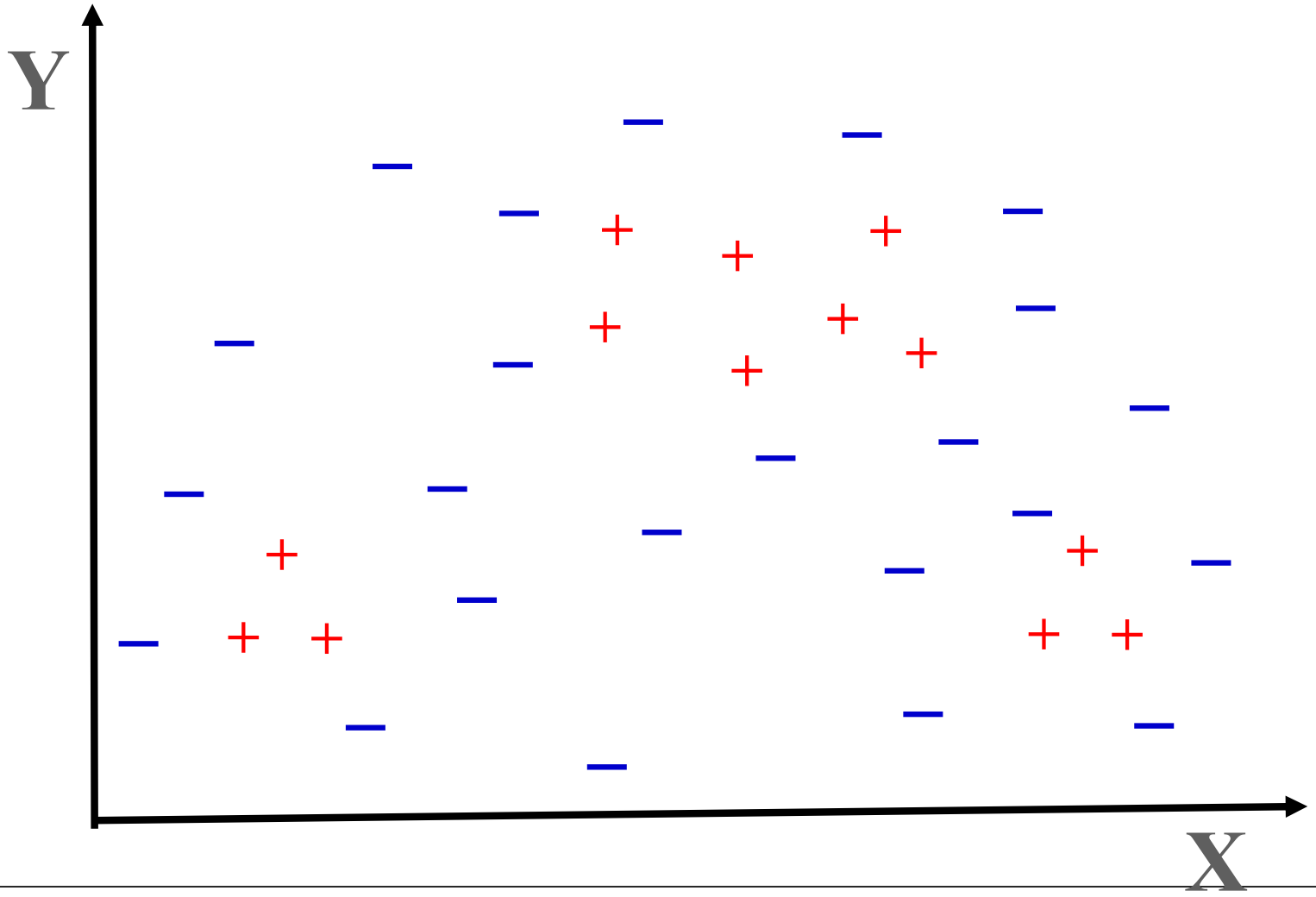
Greedy Sequential Covering Example



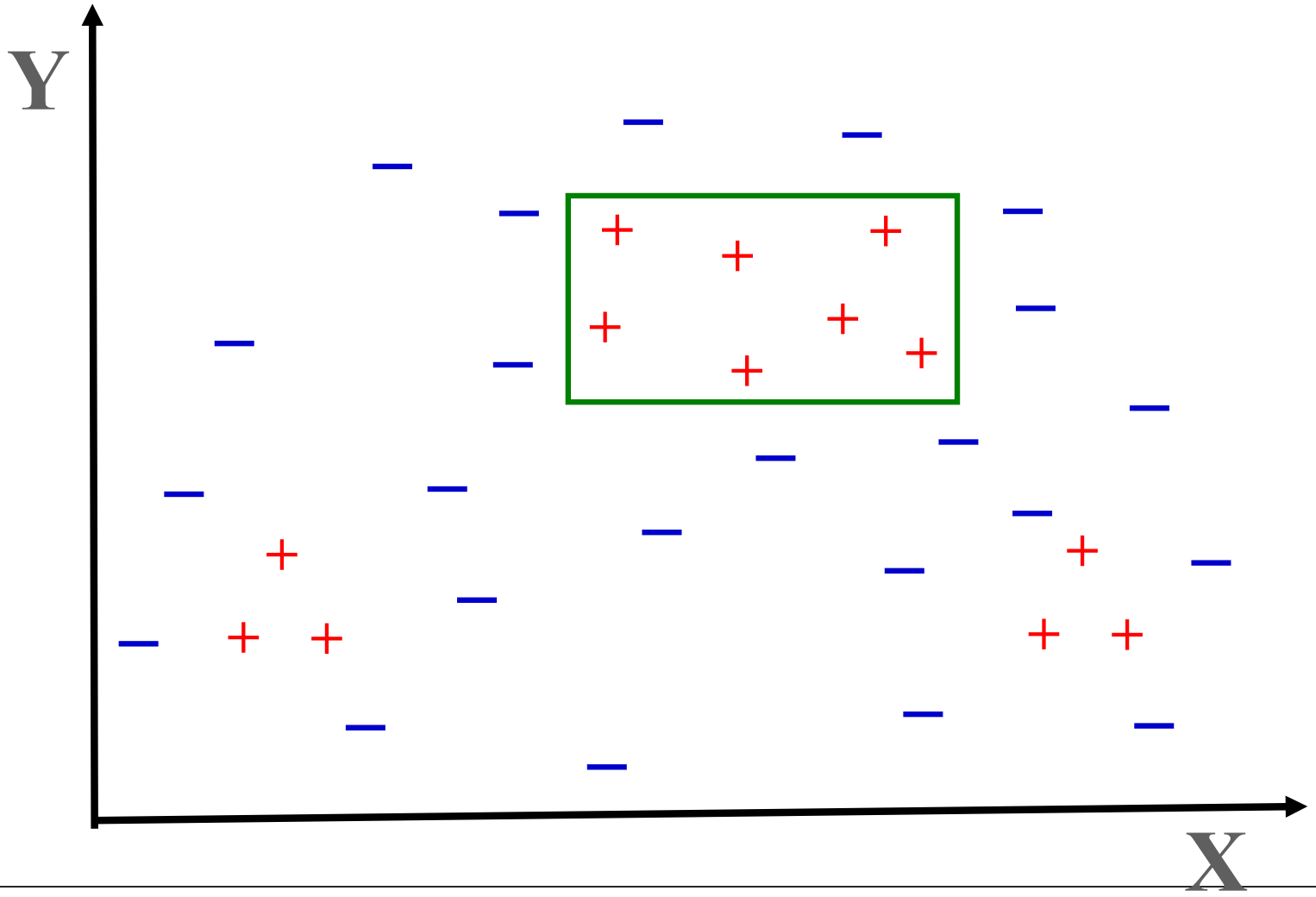
Greedy Sequential Covering Example



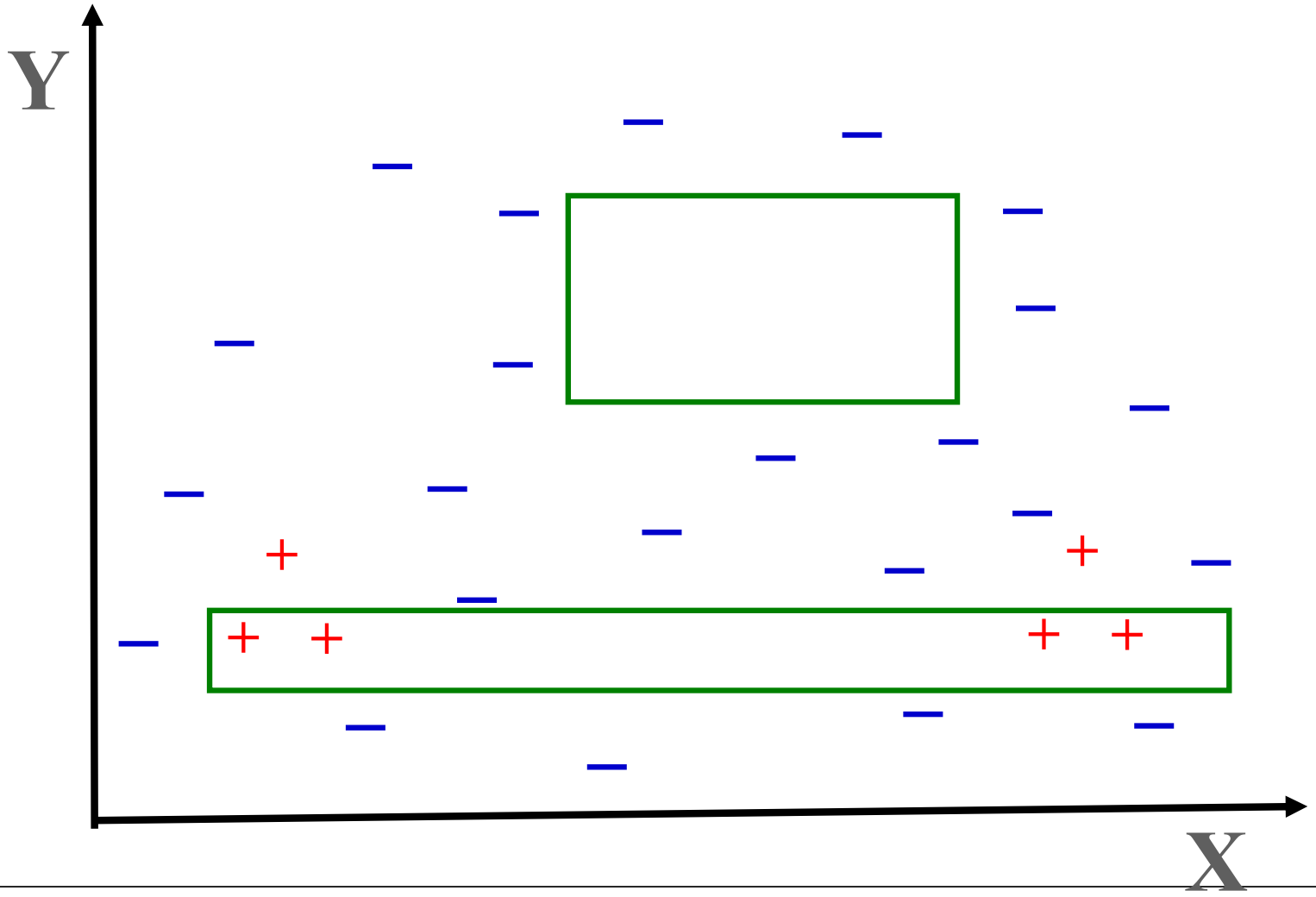
Greedy Sequential Covering Example



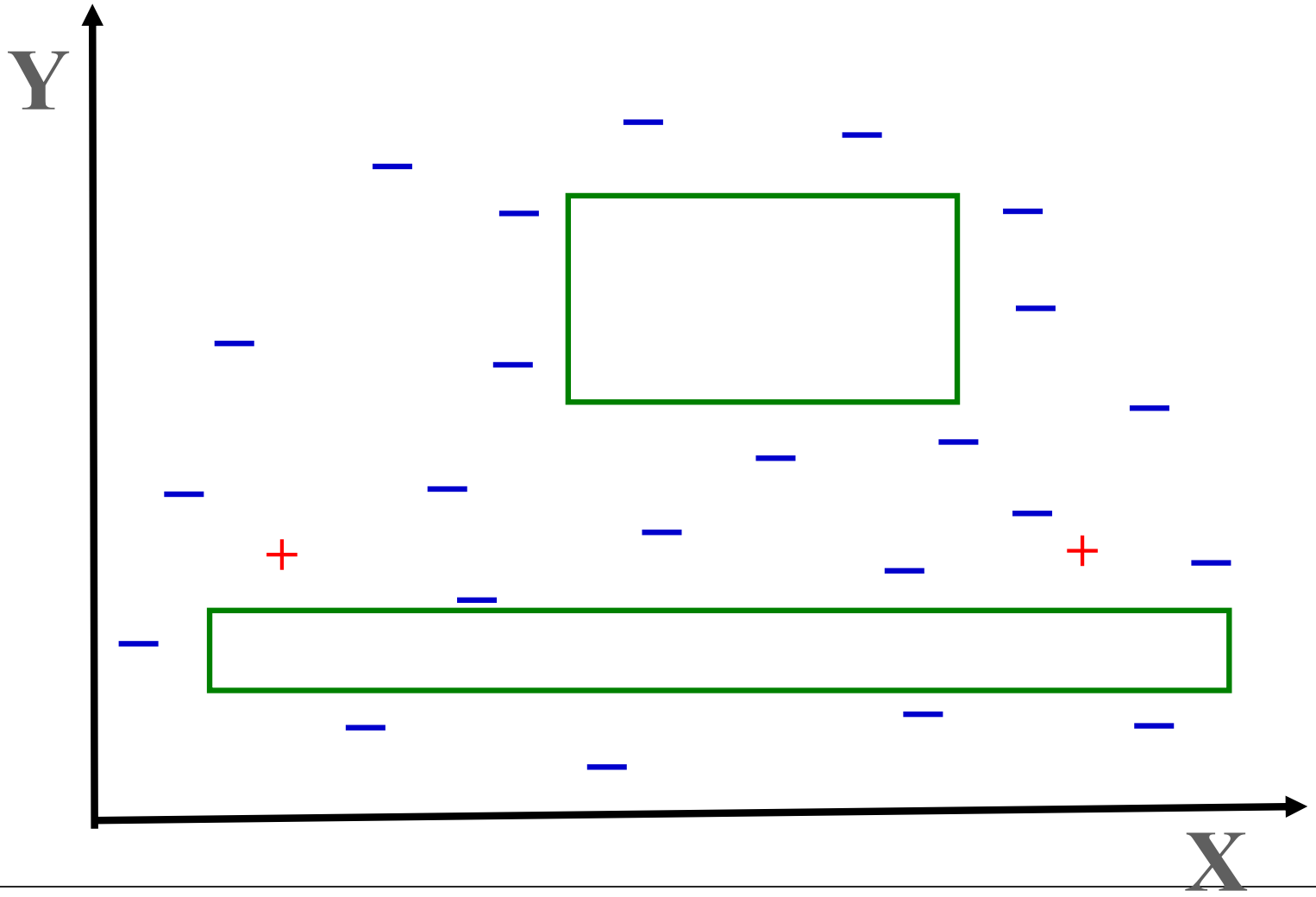
Greedy Sequential Covering Example



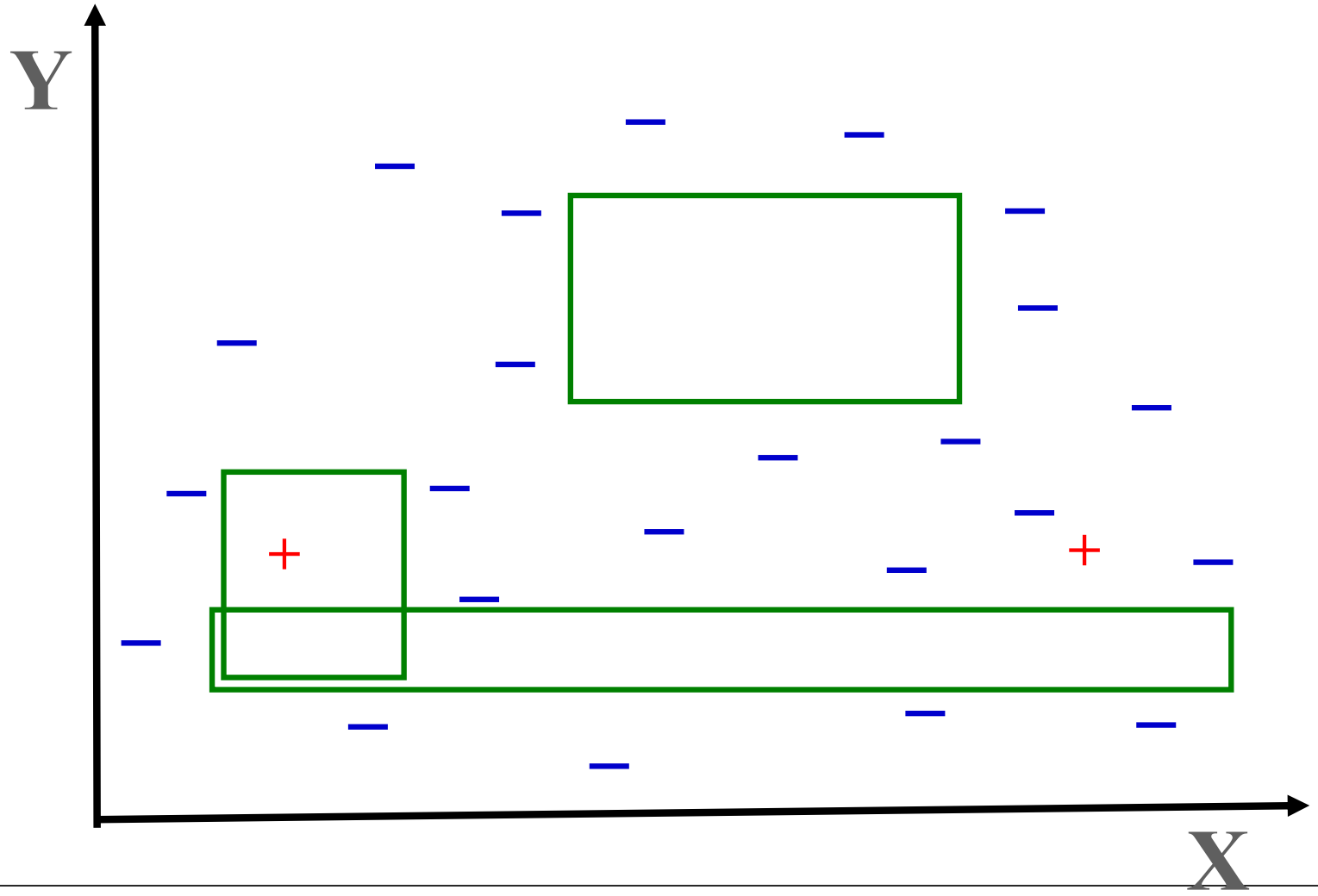
Greedy Sequential Covering Example



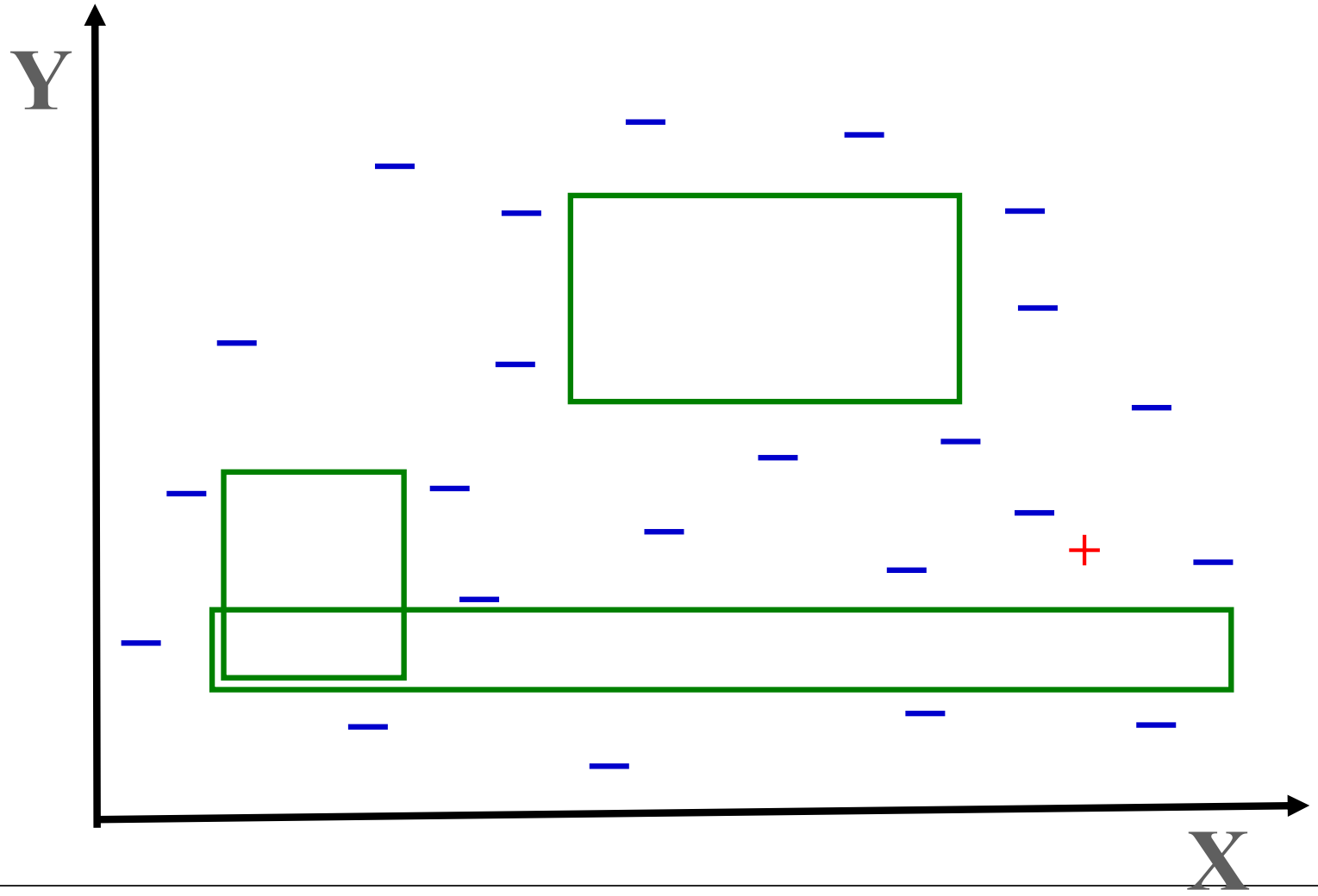
Greedy Sequential Covering Example



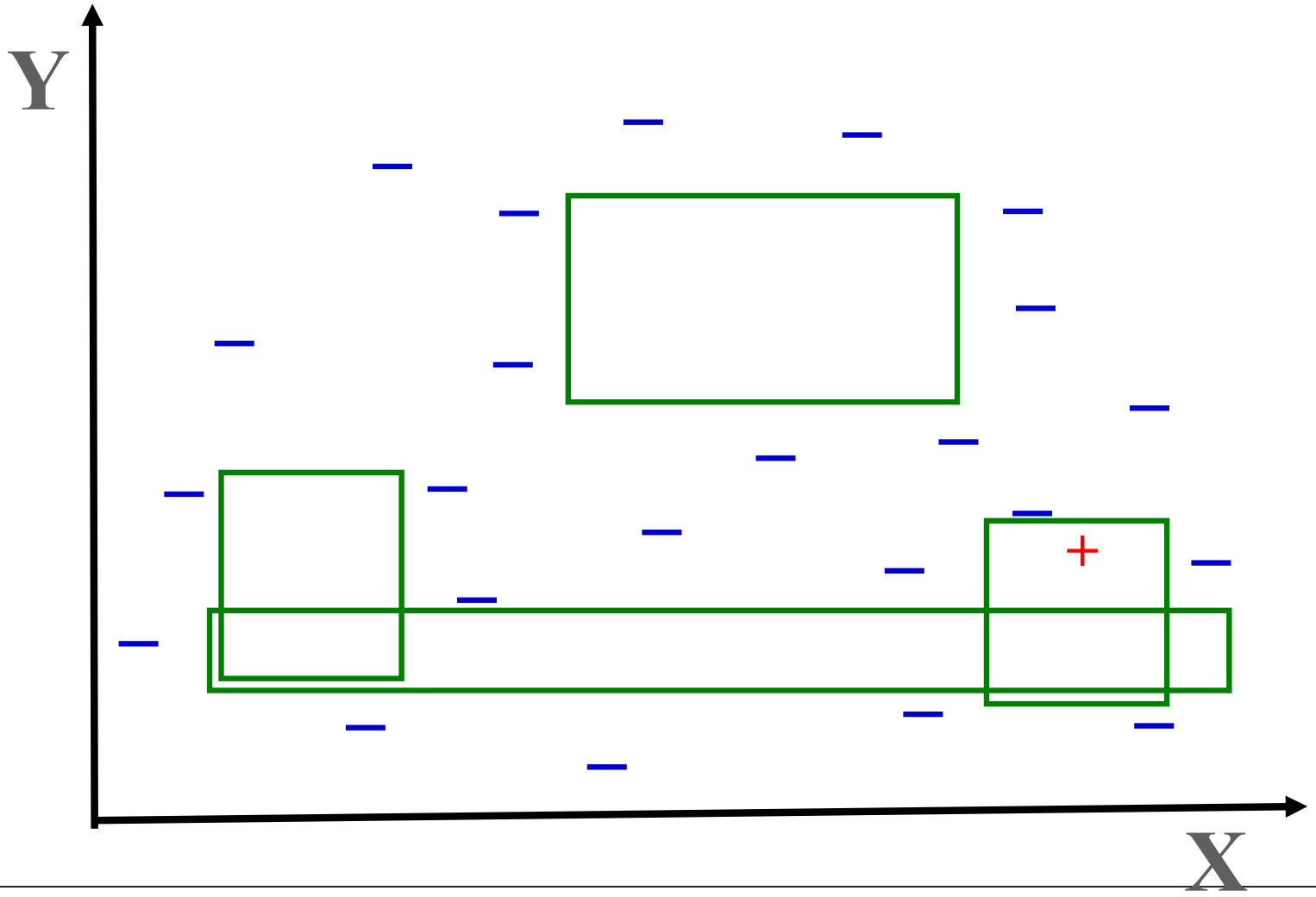
Greedy Sequential Covering Example



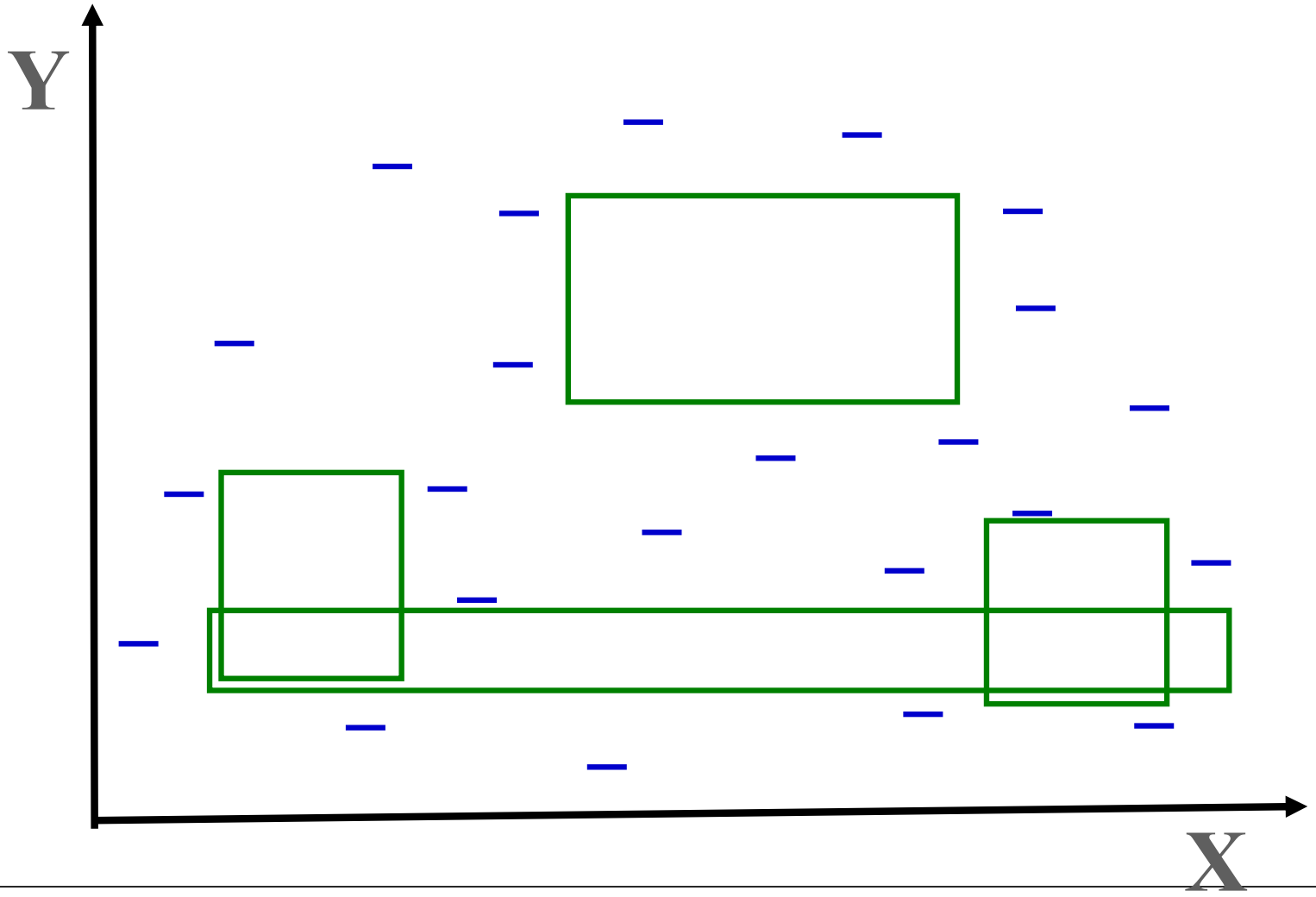
Greedy Sequential Covering Example



Greedy Sequential Covering Example



Greedy Sequential Covering Example



Strategies for Learning a Single Rule

TECHNISCHE UNIVERSITÄT DARMSTADT

- Top Down (General to Specific):
  - Start with the most-general (empty) rule.
  - Repeatedly add antecedent constraints on features that eliminate negative examples while maintaining as many positives as possible.
  - Stop when only positives are covered.
- Bottom Up (Specific to General); not covered here
  - Start with a most-specific rule (e.g. complete instance description of a random instance).
  - Repeatedly remove antecedent constraints in order to cover more positives.
  - Stop when further generalization results in covering negatives.

Illustration of learning a single Rule following FOIL

TECHNISCHE UNIVERSITÄT DARMSTADT

- FOIL is a top-down approach originally applied to first-order logic (Quinlan, 1990).
- Basic algorithm for instances with discrete-valued features:

Let A={} (set of rule antecedents)
Let N be the set of negative examples
Let P the current set of uncovered positive examples
Until N is empty do
    For every feature-value pair (literal) (F_i=V_{ij}) calculate Gain(F_i=V_{ij}, P, N)
    Pick literal, L, with highest gain.
    Add L to A.
    Remove from N any examples that do not satisfy L.
    Remove from P any examples that do not satisfy L.
Return the rule: $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow$ Positive

Foil Gain Metric

TECHNISCHE UNIVERSITÄT DARMSTADT

- We want to achieve two goals
  - Decrease coverage of negative examples N
  - Maintain coverage of as many positives examples P as possible.

Define Gain(L, P, N)

Let p be the subset of examples in P that satisfy L.

Let n be the subset of examples in N that satisfy L.

Return: $|p|*[\log_2(|p|/(|p|+|n|)) - \log_2(|P|/(|P|+|N|))]$

Sample Learning Data

TECHNISCHE UNIVERSITÄT DARMSTADT

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	big	red	circle	positive
3	small	red	triangle	negative
4	big	blue	circle	negative
5	medium	red	circle	negative

Propositional FOIL Trace

TECHNISCHE UNIVERSITÄT DARMSTADT

New Disjunct:
SIZE=BIG Gain: 0.322
SIZE=MEDIUM Gain: 0.000
SIZE=SMALL Gain: 0.322
COLOR=BLUE Gain: 0.000
COLOR=RED Gain: 0.644
COLOR=GREEN Gain: 0.000
SHAPE=SQUARE Gain: 0.000
SHAPE=TRIANGLE Gain: 0.000
SHAPE=CIRCLE Gain: 0.644
Best feature: COLOR=RED

SIZE=BIG Gain: 1.000
SIZE=MEDIUM Gain: 0.000
SIZE=SMALL Gain: 0.000
SHAPE=SQUARE Gain: 0.000
SHAPE=TRIANGLE Gain: 0.000
SHAPE=CIRCLE Gain: 0.830
Best feature: SIZE=BIG
Learned disjunct: COLOR=RED & SIZE=BIG

Propositional FOIL Trace

TECHNISCHE UNIVERSITÄT DARMSTADT

New Disjunct:
SIZE=BIG Gain: 0.000
SIZE=MEDIUM Gain: 0.000
SIZE=SMALL Gain: 1.000
COLOR=BLUE Gain: 0.000
COLOR=RED Gain: 0.415
COLOR=GREEN Gain: 0.000
SHAPE=SQUARE Gain: 0.000
SHAPE=TRIANGLE Gain: 0.000
SHAPE=CIRCLE Gain: 0.415
Best feature: SIZE=SMALL

COLOR=BLUE Gain: 0.000
COLOR=RED Gain: 0.000
COLOR=GREEN Gain: 0.000
SHAPE=SQUARE Gain: 0.000
SHAPE=TRIANGLE Gain: 0.000
SHAPE=CIRCLE Gain: 1.000
Best feature: SHAPE=CIRCLE
Learned disjunct: SIZE=SMALL & SHAPE=CIRCLE
Final Definition: COLOR=RED & SIZE=BIG v SIZE=SMALL & SHAPE=CIRCLE

Rule Learning vs. Knowledge Engineering

TECHNISCHE UNIVERSITÄT DARMSTADT

- An influential experiment with an early rule-learning method (AQ) by Michalski (1980) compared results to knowledge engineering (acquiring rules by interviewing experts).
- People known for not being able to articulate their knowledge well.
- Knowledge engineered rules:
  - Weights associated with each feature in a rule
  - Method for summing evidence similar to *certainty factors*.
  - No explicit disjunction
- Data for induction:
  - Examples of 15 soybean plant diseases described using 35 nominal and discrete ordered features, 630 total examples.
  - 290 “best” (diverse) training examples selected for training. Remainder used for testing

"Soft" Interpretation of Learned Rules
(A first hint that probabilities are important)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Certainty of match calculated for each category.
- Scoring method:
  - Literals: 1 if match, -1 if not
  - Terms (conjunctions in antecedent): Average of literal scores.
  - DNF (disjunction of rules): Probabilistic sum: $c_1 + c_2 - c_1 * c_2$

- Sample score for instance $A \wedge B \wedge \neg C \wedge D \wedge \neg E \wedge F$

  $A \wedge B \wedge C \rightarrow P$ $\quad (1 + 1 + -1)/3 = 0.333$

  $D \wedge E \wedge F \rightarrow P$ $\quad (1 + -1 + 1)/3 = 0.333$

  Total score for P: $\quad 0.333 + 0.333 - 0.333 * 0.333 = 0.555$

- Threshold of 0.8 certainty to include in possible diagnosis set.

Experimental Results

TECHNISCHE UNIVERSITÄT DARMSTADT

- Rule construction time:
  - Human: 45 hours of expert consultation
  - AQ11: 4.5 minutes training on IBM 360/75
    - What doesn't this account for?
- Test Accuracy:

	1st choice correct	Some choice correct	Number of diagnoses
MACHINE	**97.6%**	100.0%	2.64
HUMAN	**71.8%**	96.9%	2.90

Why learning relational rules? Inductive Logic Programming (ILP)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Fixed feature vectors are a very limited representation of instances.
- Examples or target concept may require relational representation that includes multiple entities with relationships between them (e.g. a graph with labeled edges and nodes).
- As you know, first-order logic is a more powerful representation for handling such relational descriptions.
- Horn clauses (i.e. if-then rules in predicate logic, Prolog programs) are a useful restriction on full first-order logic that allows decidable inference.
- Allows learning programs from sample I/O pairs.

Inductive Logic Programming (ILP)

=

Inductive Learning ∩ Logic Programming

ILP Examples

TECHNISCHE UNIVERSITÄT DARMSTADT

- Learn definitions of family relationships given data for primitive types and relations.

```
uncle(A,B) :- brother(A,C), parent(C,B).
uncle(A,B) :- husband(A,C), sister(C,D), parent(D,B).
```

- Learn recursive list programs from I/O pairs.

```
member(X,[X | Y]).
member(X, [Y | Z]) :- member(X,Z).

append([],L,L).
append([X|L1],L2,[X|L12]):-append(L1,L2,L12).
```


Introduction to ILP – Basic example

TECHNISCHE UNIVERSITÄT DARMSTADT

- Imagine learning about the relationships between people in your close family circle
- You have been told that your grandfather is the father of one of your parents, but do not yet know what a parent is
- You might have the following beliefs (**B**):

  grandfather(X, Y) ← father(X, Z), parent(Z, Y)
  father(henry, jane) ←
  mother(jane, john) ←
  mother(jane, alice) ←

- You are now given the following positive examples concerning the relationships between particular grandfathers and their grandchildren ($\mathbf{E}^+$):

  grandfather(henry, john) ←
  grandfather(henry, alice) ←

Introduction – Basic example

TECHNISCHE UNIVERSITÄT DARMSTADT

- You might be told in addition that the following relationships do not hold (negative examples) ($\mathbf{E}^-$)

  ← *grandfather(john, henry)*

  ← *grandfather(alice, john)*

- Believing **B** and faced with examples $\mathbf{E}^+$ and $\mathbf{E}^-$ you might guess the following hypothesis $\mathbf{H_1} \in \mathbf{H}$

  parent(X, Y) ← mother(X, Y)

- Here, **H** is the set of hypotheses and contains an arbitrary number of individual speculations that fit the background knowledge and examples

- Several conditions have to be fulfilled by a hypothesis. Those conditions are related to completeness and consistency with respect to the background knowledge and examples

Introduction – Basic example

TECHNISCHE UNIVERSITÄT DARMSTADT

Consistency:

- First, we must check that our problem has a solution:

$$B \cup E^- \not\models \square \text{ (}\textit{prior satisfiability}\text{)}$$

  - If one of the negative examples can be proved to be true from the background information alone, then any hypothesis we find will not be able to compensate for this. The problem is not satisfiable.
- B and H are consistent with E^-:

$$B \cup H \cup E^- \not\models \square \text{ (}\textit{posterior satisfiability}\text{)}$$

  - After adding a hypothesis it should still not be possible to prove a negative example.

Completeness:

- However, H allows us to explain E^+ relative to B:

$$B \cup H \models E^+ \text{ (}\textit{posterior sufficiency}\text{)}$$

  - This means that H should fit the positive examples given.

Model Theory – Normal Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

The problem of inductive inference:

Given is background (prior) knowledge B and evidence $E = E^+ \cup E^-$ consists of positive E^+ and negative evidence E^-,
find a hypothesis H such that the following conditions hold:

Prior Satisfiability: $B \cup E^- \not\models \square$

Posterior Satisfiability: $B \cup H \cup E^- \not\models \square$

Prior Necessity: $B \not\models E^+$

Posterior Sufficiency: $B \cup H \models E^+$

- The Sufficiency criterion is sometimes named completeness with regard to positive evidence
- The Posterior Satisfiability criterion is also known as consistency with the negative evidence
- In this general setting, background-theory, examples, and hypotheses can be any (well-formed) formula

Model Theory – Definite Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

- In most ILP practical systems background theory and hypotheses are restricted to being definite clauses
  - Clause: A disjunction of literals
  - Horn Clause: A clause with at most one positive literal
  - Definite Clause: A Horn clause with **exactly** one positive literal

$$\neg p \vee \neg q \vee \cdots \vee \neg t \vee u$$

- This setting has the advantage that a definite clause theory T has a unique minimal Herbrand model $M^+(T)$
  - Any logical formulae is either true or false in this minimal model (all formulae are decidable and the Closed World Assumption holds)

Model Theory – Definite Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

The definite semantics again require a set of conditions to hold

Consistency:

Prior Satisfiability: all e *in* E^- are false in $M^+(B)$

- Negative evidence should not be part of the minimal model

Posterior Satisfiability: all e *in* E^- are false in $M^+(B \cup H)$

- Negative evidence should not be supported by our hypotheses

Completeness

Prior Necessity: some e *in* E^+ are false in $M^+(B)$

- If all positive examples are already true in the minimal model of the background knowledge, then no hypothesis we derive will add useful information

Posterior Sufficiency: all e *in* E^+ are true in $M^+(B \cup H)$

- All positive examples are true (explained by the hypothesis) in the minimal model of the background theory and the hypothesis

Model Theory – Definite Semantics

TECHNISCHE UNIVERSITÄT DARMSTADT

- An additional restriction in addition to those of the definite semantics is to only allow true and false ground facts as examples (evidence)
- This is called the example setting
  - The example setting is the main setting employed by ILP systems
  - Only allows factual and not causal evidence (which usually captures more knowledge)

- Example:
  - B: *grandfather(X, Y) ← father(X, Z), parent(Z, Y)*
    father(henry, jane) ←
    etc.
  - E: *grandfather(henry, john) ←*
    grandfather(henry, alice) ←

Not allowed in example setting

← grandfather(X, X) — **Not allowed in definite semantics**

grandfather(henry, john) ← father(henry, jane), mother(jane, john)

Model Theory – Non-monotonic Semantics (Learning from Interpretations)

- In the nonmonotonic setting:
  - The background theory is a set of definite clauses
  - The evidence is empty
    - The positive evidence is considered part of the background theory
    - The negative evidence is derived implicitly, by making the closed world assumption (realized by the minimal Herbrand model)
  - The hypotheses are sets of general clauses expressible using *the same alphabet* as the background theory

Model Theory – Non-monotonic Semantics (2)

Since only positive evidence is present, it is assumed to be part of the background theory:

$$B' = B \cup E$$

The following conditions should hold for H and B':

- Validity: all *h in H* are true in $M^+(B')$
  - All clauses belonging to a hypothesis hold in the database B, i.e. that they are true properties of the data
- Completeness: if general clause *g* is true in $M^+(B')$ then $H \not\models g$
  - All information that is valid in the minimal model of B' should follow from the hypothesis
- Additionally the following ***can*** be a requirement:
  - Minimality: there is no proper subset *G* of *H* which is valid and complete
    - The hypothesis should not contain redundant clauses

Model Theory – Non-monotonic Semantics (3)

- Example for B (definite clauses):

  male(luc) ←

  female(lieve) ←

  human(lieve) ←

  human(luc) ←

- A possible solution is then H (a set of general clauses):

  ← female(X), male(X)

  human(X) ← male(X)

  human(X) ← female(X)

  female(X), male(X) ← human(X)

ILP as a Search Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

- ILP can be seen as a search problem - this view follows immediately from the model theory of ILP
  - In ILP there is a space of candidate solutions, i.e. the set of hypotheses, and an acceptance criterion characterizing solutions to an ILP problem
- Question: how can the space of possible solutions be structured in order to allow for pruning of the search?
  - The search space is typically structured by means of the dual notions of generalisation and specialisation
    - Generalisation corresponds to induction
    - Specialisation to deduction
    - Induction is viewed here as the inverse of deduction

Specialisation and Generalisation Rules

- A hypothesis G is more general than a hypothesis S if and only if G ⊨ S
  - S is also said to be more specific than G.
- In search algorithms, the notions of generalisation and specialisation are incorporated using inductive and deductive inference rules:
  - A deductive inference rule *r* maps a conjunction of clauses G onto a conjunction of clauses S such that G ⊨ S
    - *r* is called a specialisation rule
  - An inductive inference rule *r* maps a conjunction of clauses S onto a conjunction of clauses G such that G ⊨ S
    - *r* is called a generalisation rule

Pruning the search space

TECHNISCHE UNIVERSITÄT DARMSTADT

- Generalisation and specialisation form the basis for pruning the search space; this is because:
  - When $B \cup H \not\models e$, where $e \in E^+$, B is the background theory, H is the hypothesis, then none of the specialisations H' of H will imply the evidence
    - They can therefore be pruned from the search.
  - When $B \cup H \cup \{e\} \models \square$, where $e \in E^-$, B is the background theory, H is the hypothesis, then all generalisations H' of H will also be inconsistent with $B \cup E$
    - We can again drop them

A vanilla ILP Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- Given the key ideas of ILP as search a generic ILP system can be defined as:

$QH :=$ **Initialize**
repeat
  Delete H *from* QH
  Choose *the inference rules* $r_1, ..., r_k \in$ **R** *to be applied to* H
  Apply the rules $r_1, ..., r_k$ *to* H *to yield* $H_1, H_2, ..., H_n$
  Add $H_1, ..., H_n$ *to* QH
  Prune QH
until **stop-criterion**(QH) *satisfied*

- The algorithm works as follows:
  - It keeps track of a queue of candidate hypotheses QH
  - It repeatedly deletes a hypothesis H from the queue and expands that hypotheses using inference rules; the expanded hypotheses are then added to the queue of hypotheses QH, which may be pruned to discard unpromising hypotheses from further consideration
  - This process continues until the stopcriterion is satisfied

Some Concrete ILP Systems

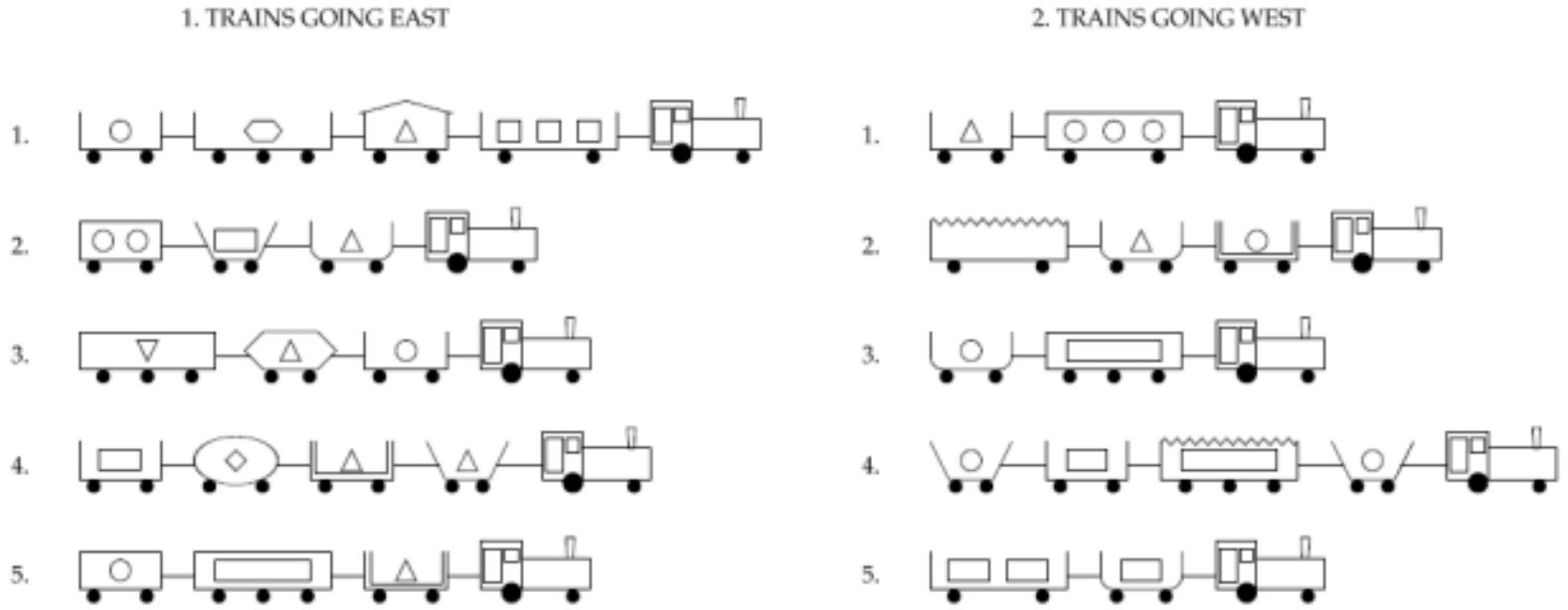
TECHNISCHE UNIVERSITÄT DARMSTADT

- Top-Down:
  - FOIL (Quinlan, 1990)
- Bottom-Up:
  - CIGOL (Muggleton & Buntine, 1988)
  - GOLEM (Muggleton, 1990)
- Hybrid:
  - CHILLIN (Mooney & Zelle, 1994)
  - PROGOL (Muggleton, 1995)
  - ALEPH (Srinivasan, 2000)
- Non-monotonic:
  - Claudien (De Raedt)
  - TILDE (Blockeel, De Raedt)

Michalski's train problem

TECHNISCHE UNIVERSITÄT DARMSTADT

- Assume ten railway trains: five are travelling east and five are travelling west; each train comprises a locomotive pulling wagons; whether a particular train is travelling towards the east or towards the west is determined by some properties of that train



- The learning task: determine what governs which kinds of trains are Eastbound and which kinds are Westbound

Michalski's train problem (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Michalski's train problem can be viewed as a classification task: the aim is to generate a classifier (theory) which can classify unseen trains as either Eastbound or Westbound
- The following knowledge about each car can be extracted: which train it is part of, its shape, how many wheels it has, whether it is open (i.e. has no roof) or closed, whether it is long or short, the shape of the things the car is loaded with. In addition, for each pair of connected wagons, knowledge of which one is in front of the other can be extracted.

Michalski's train problem (3)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Examples of Eastbound trains
  - Positive examples:

    eastbound(east1).
    eastbound(east2).
    eastbound(east3).
    eastbound(east4).
    eastbound(east5).
  - Negative examples:

    eastbound(west6).
    eastbound(west7).
    eastbound(west8).
    eastbound(west9).
    eastbound(west10).

Michalski's train problem (4)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Background knowledge for train *east1*. Cars are uniquely identified by constants of the form *car_xy*, where *x* is number of the train to which the car belongs and *y* is the position of the car in that train. For example *car_12* refers to the second car behind the locomotive in the first train
  - short(car_12). short(car_14).
  - long(car_11). long(car_13).
  - closed(car_12).
  - open(car_11). open(car_13). open(car_14).
  - infront(east1,car_11). infront(car_11,car_12).
  - infront(car_12,car_13). infront(car_13,car_14).
  - shape(car_11,rectangle). shape(car_12,rectangle).
  - shape(car_13,rectangle). shape(car_14,rectangle).
  - load(car_11,rectangle,3). load(car_12,triangle,1).
  - load(car_13,hexagon,1). load(car_14,circle,1).
  - wheels(car_11,2). wheels(car_12,2).
  - wheels(car_13,3). wheels(car_14,2).
  - has_car(east1,car_11). has_car(east1,car_12).
  - has_car(east1,car_13). has_car(east1,car_14).

Michalski's train problem (5)

TECHNISCHE UNIVERSITÄT DARMSTADT

- An ILP systems could generate the following hypothesis:

  eastbound(A) ← has_car(A,B), not(open(B)), not(long(B)).

  i.e. a train A is eastbound if it has a car which is both not open and not long.
- Other generated hypotheses could be:
  - If a train has a short closed car, then it is Eastbound and otherwise Westbound
  - If a train has two cars, or has a car with a corrugated roof, then it is Westbound and otherwise Eastbound
  - If a train has more than two different kinds of load, then it is Eastbound and otherwise Westbound
  - For each train add up the total number of sides of loads (taking a circle to have one side); if the answer is a divisor of 60 then the train is Westbound andotherwise Eastbound

FOIL (First-Order Inductive Logic) … again

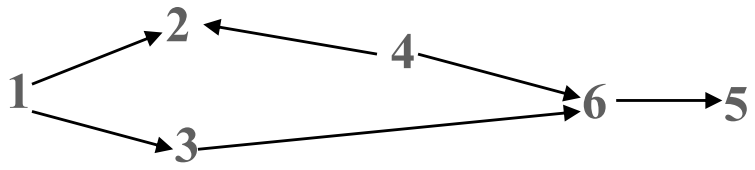
- Top-down sequential covering algorithm "upgraded" to learn Prolog clauses, but without logical functions.
- Background knowledge must be provided extensionally.
- Initialize clause for target predicate P to
  $P(X_1,\ldots X_T)$:-.
- Possible specializations of a clause include adding all possible literals:
  - $Q_i(V_1,\ldots,V_{Ti})$
  - not($Q_i(V_1,\ldots,V_{Ti})$)
  - $X_i = X_j$
  - not($X_i = X_j$)

  where X's are "bound" variables already in the existing clause; at least one of $V_1,\ldots,V_{Ti}$ is a bound variable, others can be new.
- Allow recursive literals $P(V_1,\ldots,V_T)$ if they do not cause an infinite regress.
- Handle alternative possible values of new intermediate variables by maintaining examples as tuples of all variable values.

FOIL Training Data

- For learning a recursive definition, the positive set must consist of *all* tuples of constants that satisfy the target predicate, given some fixed universe of constants.
- Background knowledge consists of complete set of tuples for each background predicate for this universe.
- Example: Consider learning a definition for the target predicate `path` for finding a path in a directed acyclic graph.

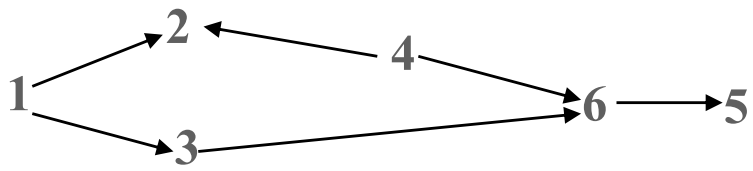
```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```



edge: {<1,2>,<1,3>,<3,6>,<4,2>,<4,6>,<6,5>}
path: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

FOIL Negative Training Data

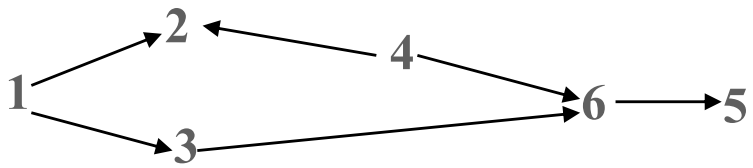
- Negative examples of target predicate can be provided directly, or generated indirectly by making a *closed world assumption*.
  - Every pair of constants <X,Y> not in positive tuples for `path` predicate.



Negative path tuples:

{<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Sample FOIL Induction



Pos: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Start with clause:

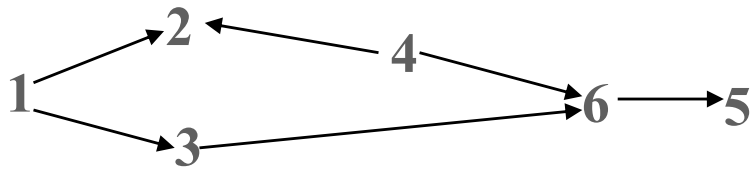
path(X,Y):-.

Possible literals to add:

edge(X,X),edge(Y,Y),edge(X,Y),edge(Y,X),edge(X,Z),
edge(Y,Z),edge(Z,X),edge(Z,Y),path(X,X),path(Y,Y),
path(X,Y),path(Y,X),path(X,Z),path(Y,Z),path(Z,X),
path(Z,Y),X=Y,

plus negations of all of these.

Sample FOIL Induction



Pos: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

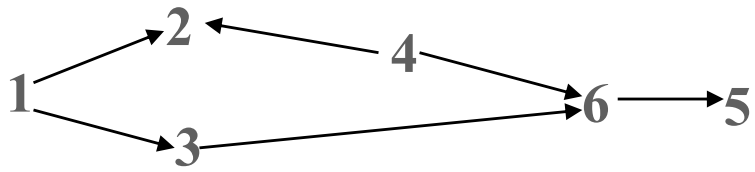
path(X,Y):- edge(X,X).

Covers 0 positive examples

Covers 6 negative examples

Not a good literal.

Sample FOIL Induction



Pos: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

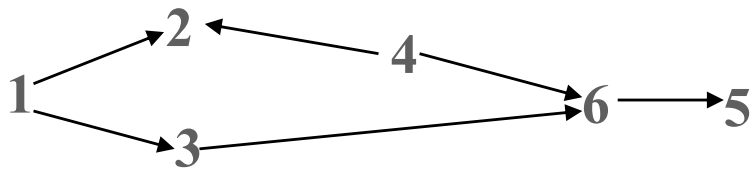
path(X,Y):- edge(X,Y).

Covers 6 positive examples

Covers 0 negative examples

Chosen as best literal. Result is base clause.

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>, <4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

path(X,Y):- edge(X,Y).

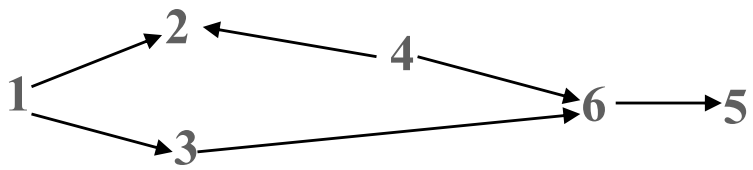
Covers 6 positive examples

Covers 0 negative examples

Chosen as best literal. Result is base clause.

Remove covered positive tuples.

Sample FOIL Induction



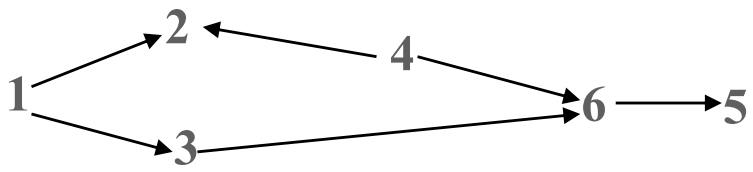
Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Start new clause

path(X,Y):-.

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

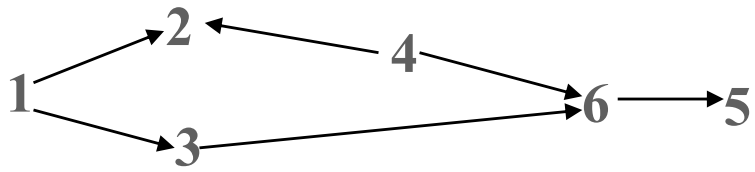
path(X,Y):- edge(X,Y).

Covers 0 positive examples

Covers 0 negative examples

Not a good literal.

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

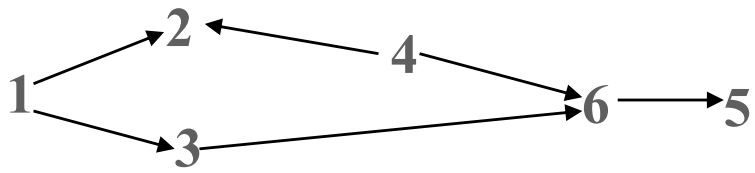
Test:
path(X,Y):- edge(X,Z).

Covers all 4 positive examples

Covers 14 of 26 negative examples

Eventually chosen as best possible literal

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,<4,5>}

Neg: {<1,1>,<1,4>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,
<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

path(X,Y):- edge(X,Z).

Covers all 4 positive examples

Covers 15 of 26 negative examples

Eventually chosen as best possible literal

Continue in this fashion, while extending the tuples by Z (all the variables)

Recursion Limitation

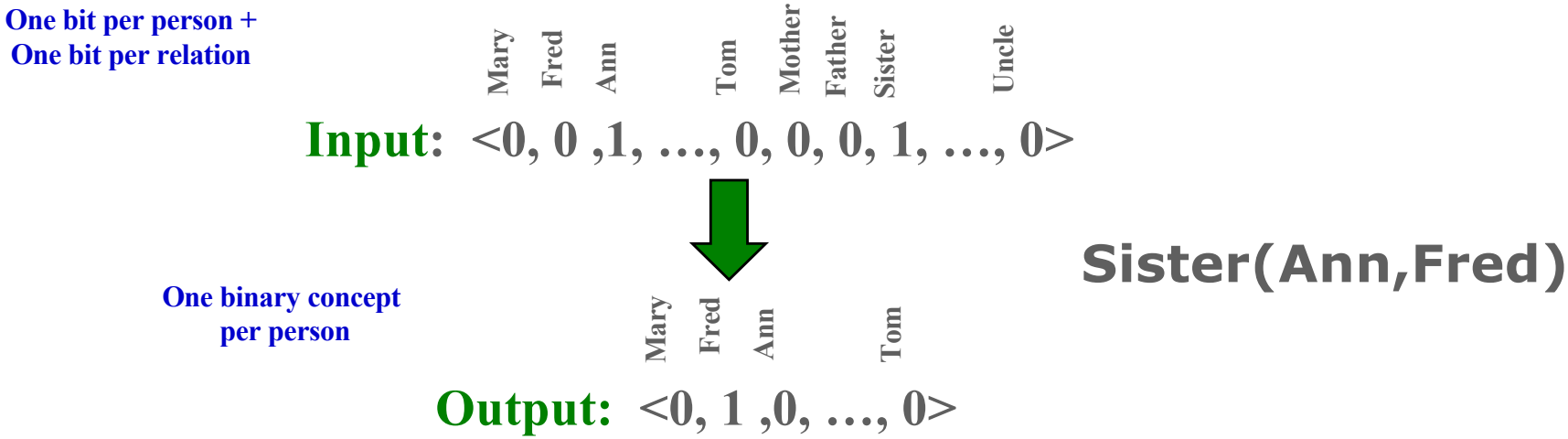
TECHNISCHE UNIVERSITÄT DARMSTADT

- Must not build a clause that results in an infinite regress.
  - `path(X,Y) :- path(X,Y).`
  - `path(X,Y) :- path(Y,X).`
- To guarantee termination of the learned clause, must "reduce" at least one argument according to some well-founded partial ordering.
- A binary predicate, *R*, is a well-founded partial ordering if the transitive closure does not contain *R*(*a*,*a*) for any constant *a*.
  - `rest(A,B)`
  - `edge(A,B)` for an acyclic graph

Learning Family Relations

TECHNISCHE UNIVERSITÄT DARMSTADT

- FOIL can learn accurate Prolog definitions of family relations such as wife, husband, mother, father, daughter, son, sister, brother, aunt, uncle, nephew and niece, given basic data on parent, spouse, and gender for a particular family.
- Produces significantly more accurate results than feature-based learners (e.g. neural nets) applied to a "flattened" ("propositionalized") and restricted version of the problem (back at that time).



Example: Learn Prolog Program for List Membership

- Target:
  - `member`:
    `(a,[a]),(b,[b]),(a,[a,b]),(b,[a,b]),…`
- Background:
  - `components`:
    `([a],a,[]),([b],b,[]),([a,b],a,[b]),`
    `([b,a],b,[a]),([a,b,c],a,[b,c]),…`
- Definition:

```
member(A,B) :- components(B,A,C).
member(A,B) :- components(B,C,D),
               member(A,D).
```


Logic Program Induction in FOIL

- FOIL has also learned
  - `append` given `components` and `null`
  - `reverse` given `append`, `components`, and `null`
  - `quicksort` given `partition`, `append`, `components`, and `null`
  - Other programs from the first few chapters of a Prolog text.
- Learning recursive programs in FOIL requires a complete set of positive examples for some constrained universe of constants, so that a recursive call can always be evaluated extensionally.
  - For lists, all lists of a limited length composed from a small set of constants (e.g. all lists up to length 3 using {a,b,c}).
  - Size of extensional background grows combinatorially.
- Negative examples usually computed using a closed-world assumption.
  - Grows combinatorially large for higher arity target predicates.
  - Can randomly sample negatives to make tractable.

More Realistic Applications

TECHNISCHE UNIVERSITÄT DARMSTADT

- Classifying chemical compounds as mutagenic (cancer causing) based on their graphical molecular structure and chemical background knowledge.
- Classifying web documents based on both the content of the page and its links to and from other pages with particular content.
  - A web page is a university faculty home page if:
    - It contains the words “Professor” and “University”, and
    - It is pointed to by a page with the word “faculty”, and
    - It points to a page with the words “course” and “exam”

FOIL Limitations

TECHNISCHE UNIVERSITÄT DARMSTADT

- Search space of literals (branching factor) can become intractable.
  - Use aspects of bottom-up search to limit search.
- Requires large extensional background definitions.
  - Use intensional background via Prolog inference.
- Hill-climbing search gets stuck at local optima and may not even find a consistent clause.
  - Use limited backtracking (beam search)
  - Include determinate literals with zero gain.
  - Use relational e.g. pathfinding.
- Requires complete examples to learn recursive definitions.
  - Use intensional interpretation of learned recursive clauses.

FOIL Limitations (cont.)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Requires a large set of closed-world negatives.
  - Exploit "output completeness" to provide "implicit" negatives.
    - `past-tense([s,i,n,g], [s,a,n,g])`
- Inability to handle logical functions.
  - Use bottom-up methods that handle functions
- Background predicates must be sufficient to construct definition, e.g. cannot learn `reverse` unless given `append`.
  - Predicate invention
    - Learn `reverse` by inventing `append`
    - Learn `sort` by inventing `insert`

Summary ILP

TECHNISCHE UNIVERSITÄT DARMSTADT

- ILP is a subfield of machine learning which uses logic programming as a uniform representation for examples, background knowledge and hypotheses
- Many existing ILP systems
  - Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesised logic program which entails all the positive and none of the negative examples
- Lots of applications of ILP
  - E.g. bioinformatics, natural language processing, engineering

What have we learnt?

TECHNISCHE UNIVERSITÄT DARMSTADT

- There are effective methods for learning symbolic rules from data using greedy sequential covering and top-down or bottom-up search.
- These methods have been extended to first-order logic to learn relational rules and recursive Prolog programs.
- Knowledge represented by rules is generally more interpretable by people, allowing human insight into what is learned and possible human approval and correction of learned knowledge.