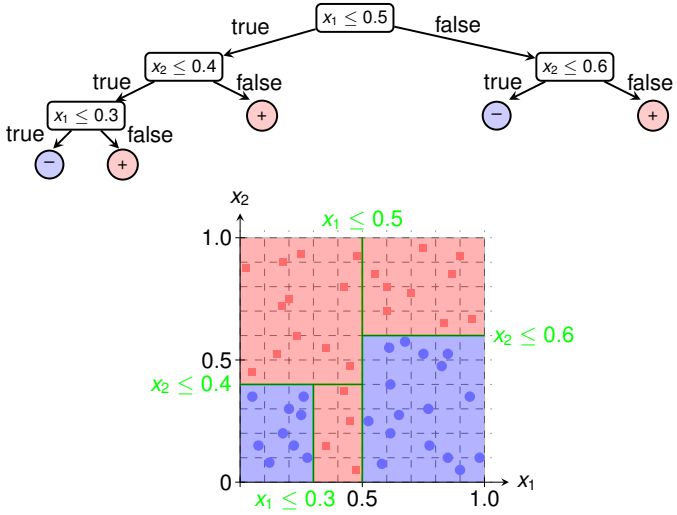


Cambridge AI+

Lecture 7: Decision Trees

Thomas Sauerwald

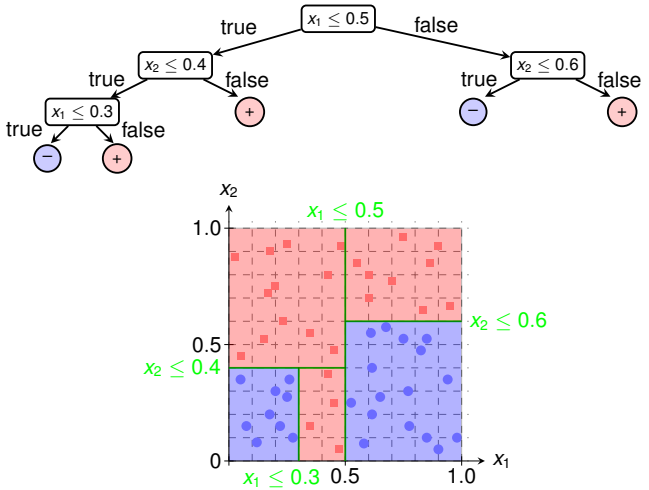
University of Cambridge, Department of Computer Science and Technology

Feb 2021

UNIVERSITY OF CAMBRIDGE

Outline

Decision Trees

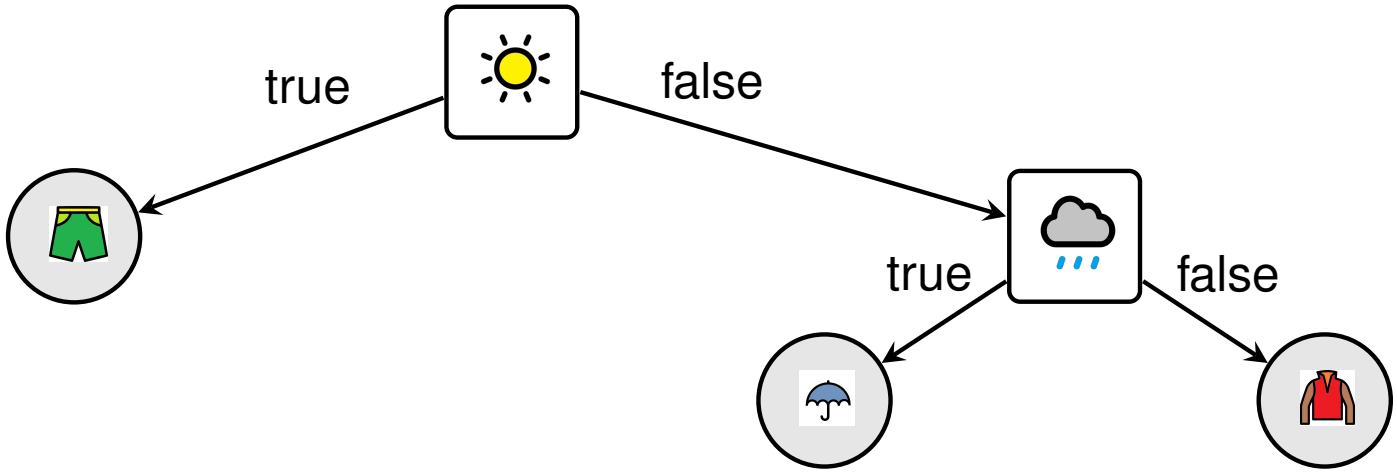


Decision Trees

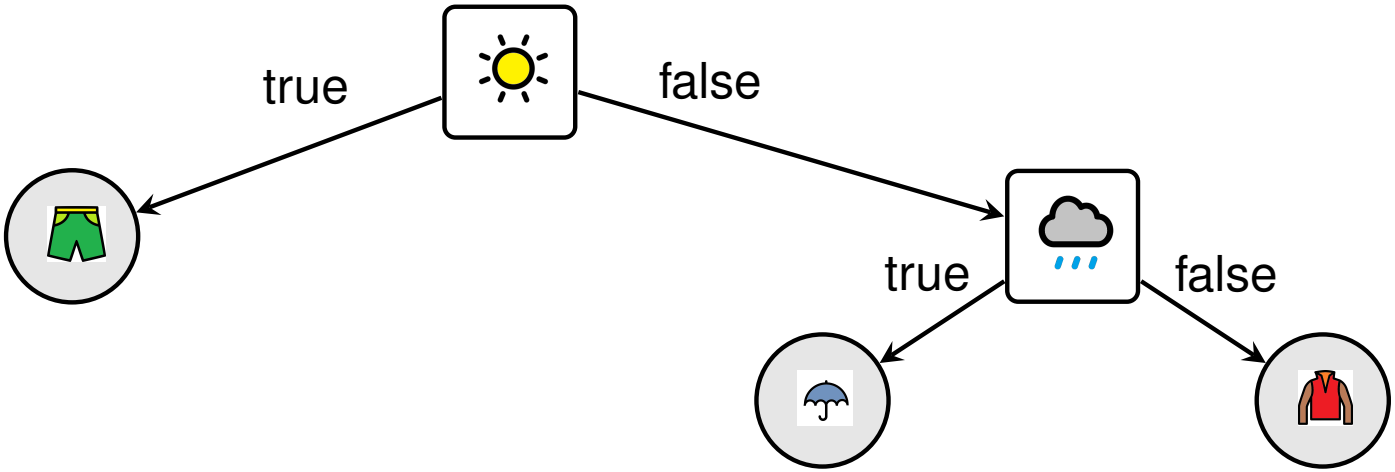
- a decision tree is a tree-structured plan with binary queries about the features in order to predict the output
- features (and labels) can be categorical or numerical:
  - categorical labels: classification trees
  - numerical labels: regression trees
- unlike kNN (or Perceptron), need a separate learning phase before any classification
- basic component of Random Forests, which employ majority vote over randomly constructed decision trees

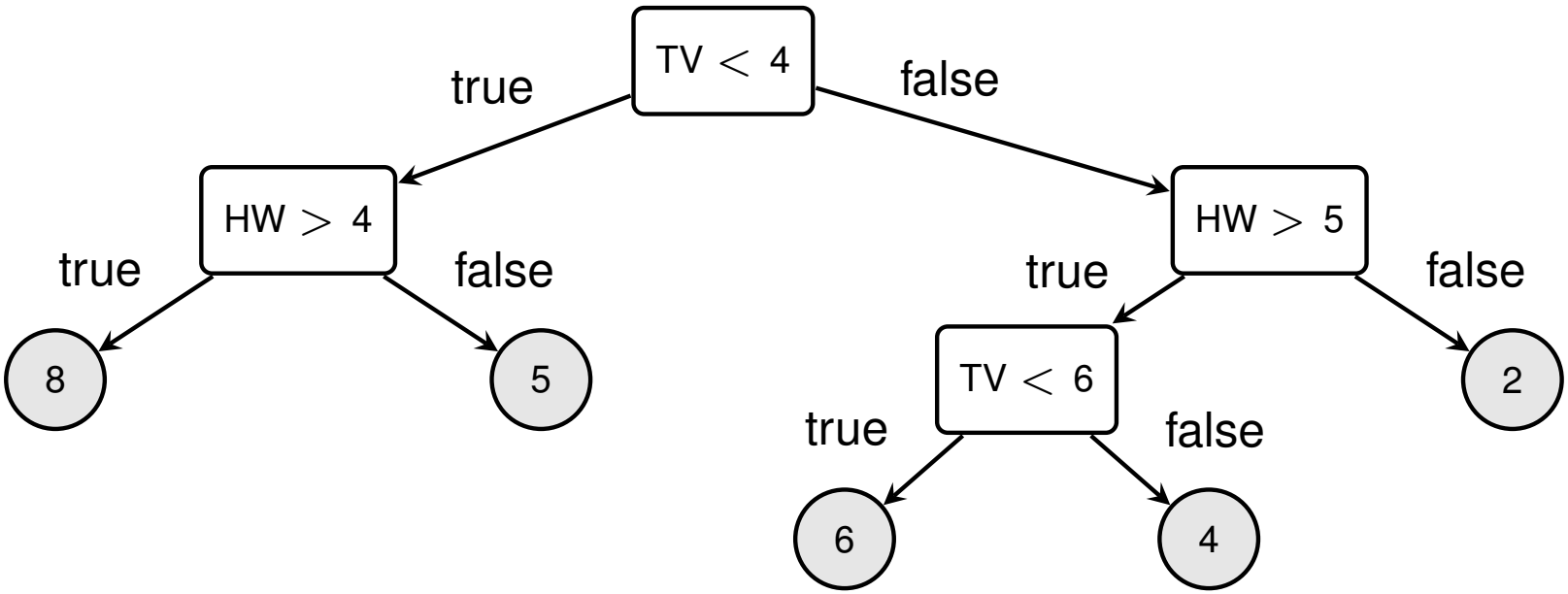
Examples: Classification and Regression using Decision Trees

Examples: Classification and Regression using Decision Trees

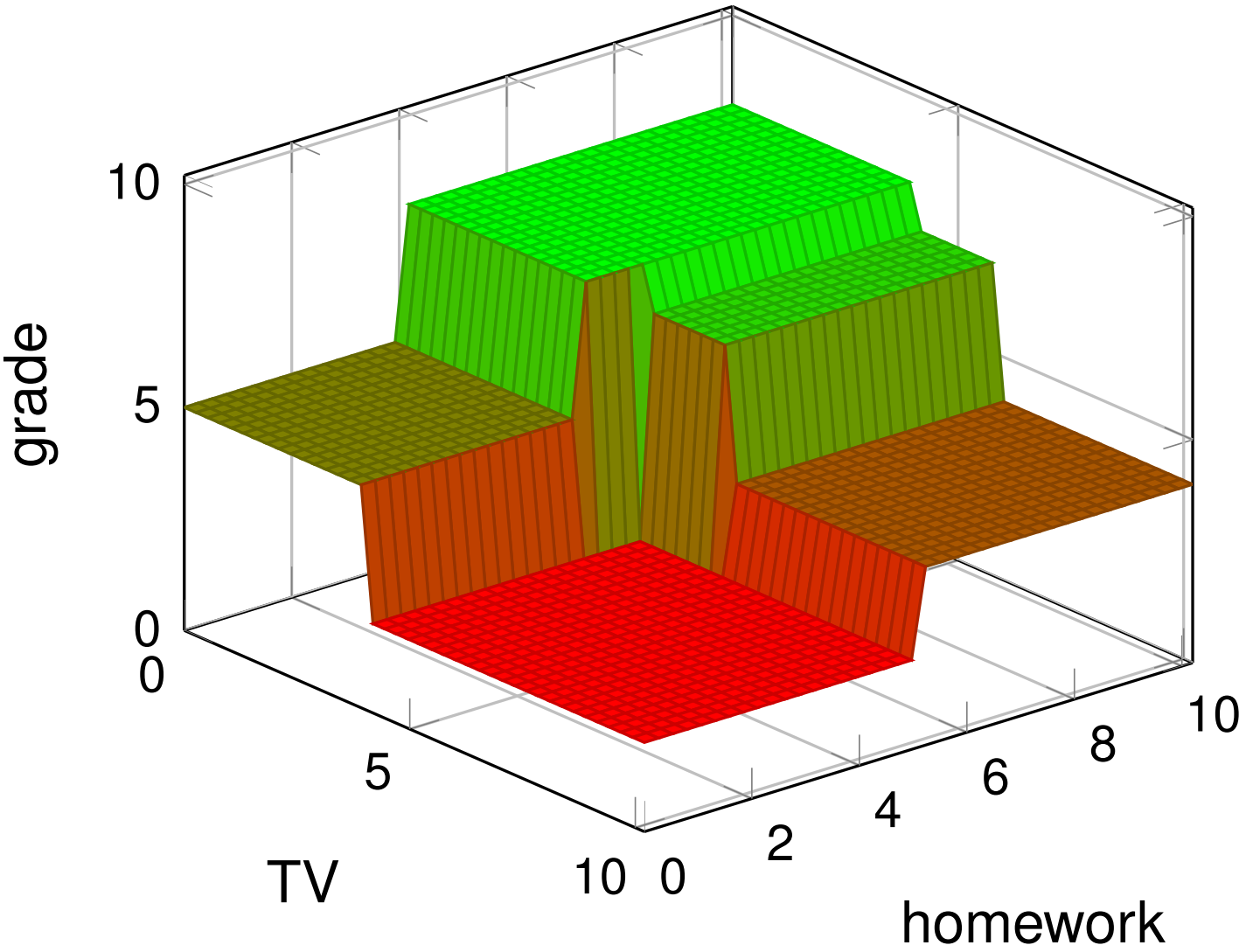


Examples: Classification and Regression using Decision Trees





Regression Tree



A Simple Binary Classification Rule

Decision Stumps

- Let x be a single feature value

A Simple Binary Classification Rule

Decision Stumps

- Let x be a single feature value
- Then find a threshold $\Theta \in \mathbb{R}$ such that either:
  - If $x \geq \Theta$, then assign it a positive label.
  - If $x < \Theta$, then assign it a negative label.

A Simple Binary Classification Rule

Decision Stumps

- Let x be a single feature value
- Then find a threshold $\Theta \in \mathbb{R}$ such that either:
  - If $x \geq \Theta$, then assign it a positive label.
  - If $x < \Theta$, then assign it a negative label.
- or:
  - If $x \geq \Theta$, then assign it a negative label.
  - If $x < \Theta$, then assign it a positive label.

A Simple Binary Classification Rule

These are the most basic types of **Decision trees**!

Decision Stumps

- Let x be a single feature value
- Then find a threshold $\Theta \in \mathbb{R}$ such that either:
  - If $x \geq \Theta$, then assign it a positive label.
  - If $x < \Theta$, then assign it a negative label.
- or:
  - If $x \geq \Theta$, then assign it a negative label.
  - If $x < \Theta$, then assign it a positive label.

A Simple Binary Classification Rule

These are the most basic types of **Decision trees**!

Decision Stumps

- Let x be a single feature value
- Then find a threshold $\Theta \in \mathbb{R}$ such that either:
  - If $x \geq \Theta$, then assign it a positive label.
  - If $x < \Theta$, then assign it a negative label.
- or:
  - If $x \geq \Theta$, then assign it a negative label.
  - If $x < \Theta$, then assign it a positive label.

We frequently apply these rules in our daily life (consciously or unconsciously!)

- I will watch this movie if the average rating is at least 4.5
- I will buy these shoes once the price drops below 100
- I will go jogging today if the temperature is above 20
- ...

Schematic Example (1/3)

- Assume we are given a data set with 5 faces and 5 non-faces, and we want to find best decision stump
- Each feature is a single number obtained by computing a difference of colours in the grey and white area (and normalising)
- This is a sub-problem arising in face-detection applications, for example in Adaptive Boosting

Schematic Example (1/3)

- Assume we are given a data set with 5 faces and 5 non-faces, and we want to find best decision stump
- Each feature is a single number obtained by computing a difference of colours in the grey and white area (and normalising)
- This is a sub-problem arising in face-detection applications, for example in Adaptive Boosting

F N F N N F F F N N

Schematic Example (1/3)

- Assume we are given a data set with 5 faces and 5 non-faces, and we want to find best decision stump
- Each feature is a single number obtained by computing a difference of colours in the grey and white area (and normalising)
- This is a sub-problem arising in face-detection applications, for example in Adaptive Boosting

F N F N N F F F N N

For simplicity, we only choose two features (kernels) in this example:

Schematic Example (1/3)

- Assume we are given a data set with 5 faces and 5 non-faces, and we want to find best decision stump
- Each feature is a single number obtained by computing a difference of colours in the grey and white area (and normalising)
- This is a sub-problem arising in face-detection applications, for example in Adaptive Boosting

F N F N N F F F N N

For simplicity, we only choose two features (kernels) in this example:

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1			+
N	Image 2			−
F	Image 3			+
N	Image 4			−
N	Image 5			−
F	Image 6			+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2			−
F	Image 3			+
N	Image 4			−
N	Image 5			−
F	Image 6			+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3			+
N	Image 4			−
N	Image 5			−
F	Image 6			+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4			−
N	Image 5			−
F	Image 6			+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5			−
F	Image 6			+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6			+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7			+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8			+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9			−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10			−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16		+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315		−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16		+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377		−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605		−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605	0.485	−
F	Image 6	0.585		+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605	0.485	−
F	Image 6	0.585	0.895	+
F	Image 7	0.355		+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605	0.485	−
F	Image 6	0.585	0.895	+
F	Image 7	0.355	0.793	+
F	Image 8	0.737		+
N	Image 9	0.86		−
N	Image 10	0.91		−

Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605	0.485	−
F	Image 6	0.585	0.895	+
F	Image 7	0.355	0.793	+
F	Image 8	0.737	0.747	+
N	Image 9	0.86		−
N	Image 10	0.91		−

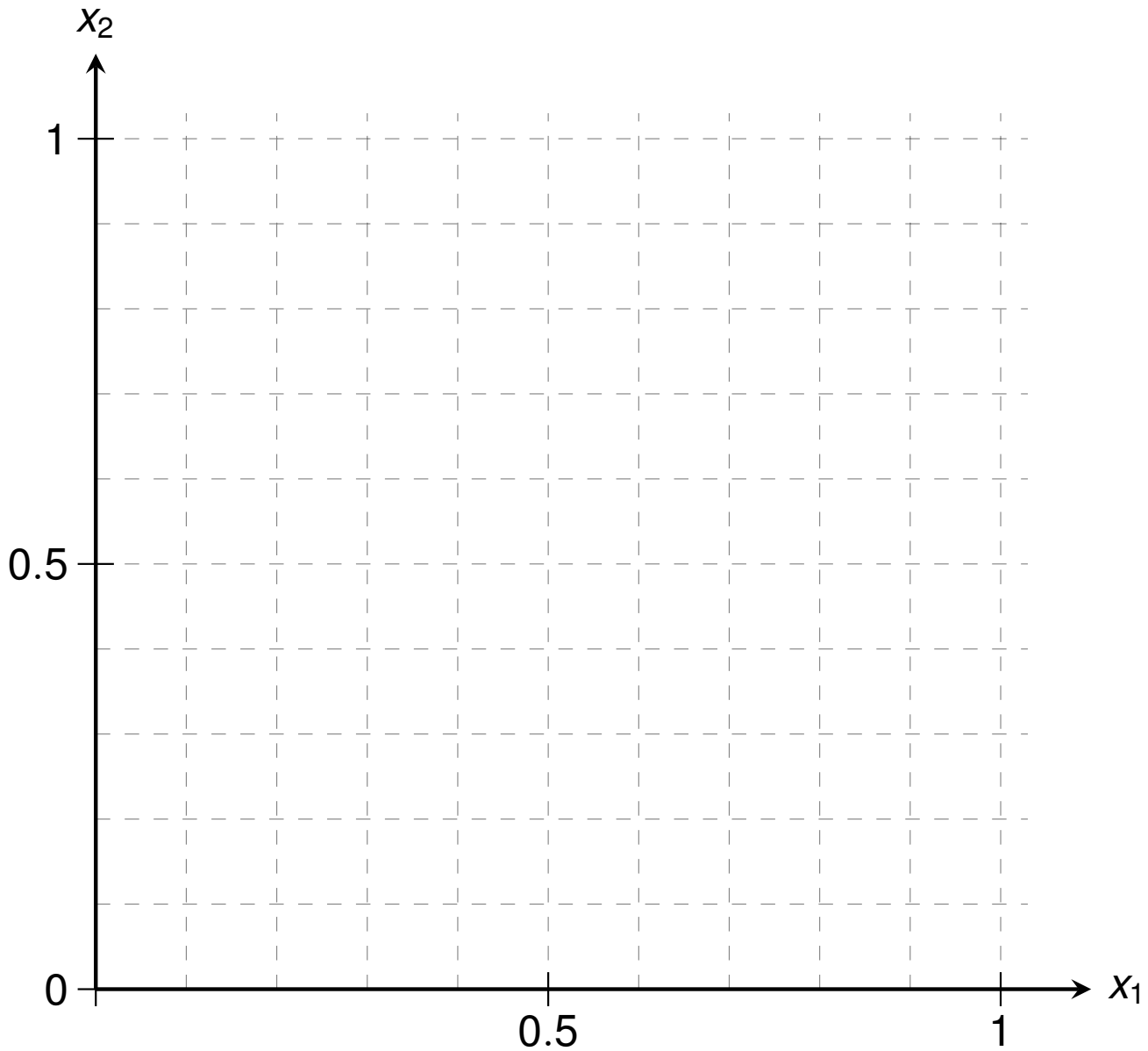
Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605	0.485	−
F	Image 6	0.585	0.895	+
F	Image 7	0.355	0.793	+
F	Image 8	0.737	0.747	+
N	Image 9	0.86	0.25	−
N	Image 10	0.91		−

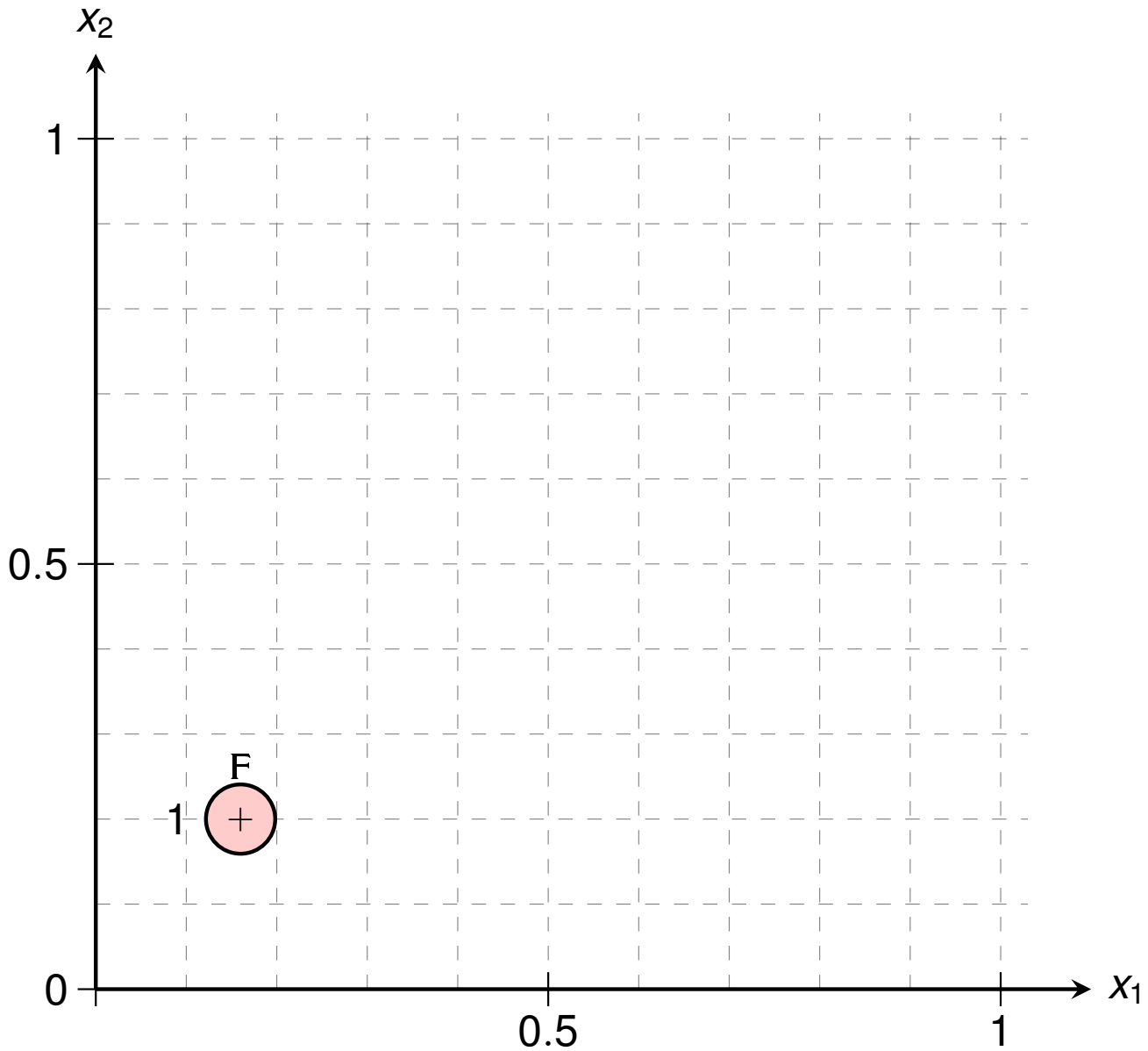
Schematic Example (2/3)

		Feature 1	Feature 2	Label
F	Image 1	0.16	0.2	+
N	Image 2	0.315	0.09	−
F	Image 3	0.16	0.53	+
N	Image 4	0.377	0.56	−
N	Image 5	0.605	0.485	−
F	Image 6	0.585	0.895	+
F	Image 7	0.355	0.793	+
F	Image 8	0.737	0.747	+
N	Image 9	0.86	0.25	−
N	Image 10	0.91	0.84	−

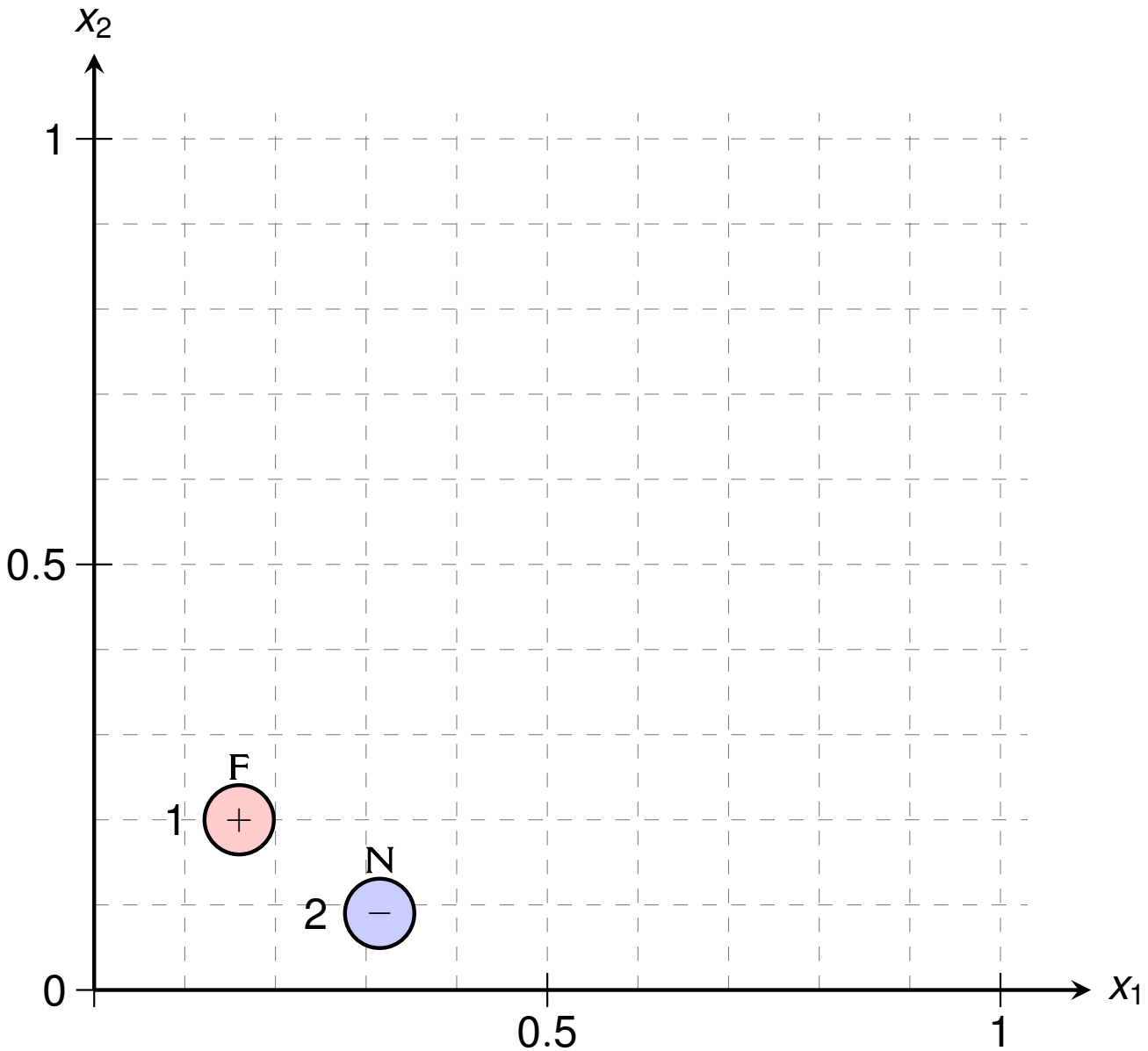
Schematic Example (3/3)



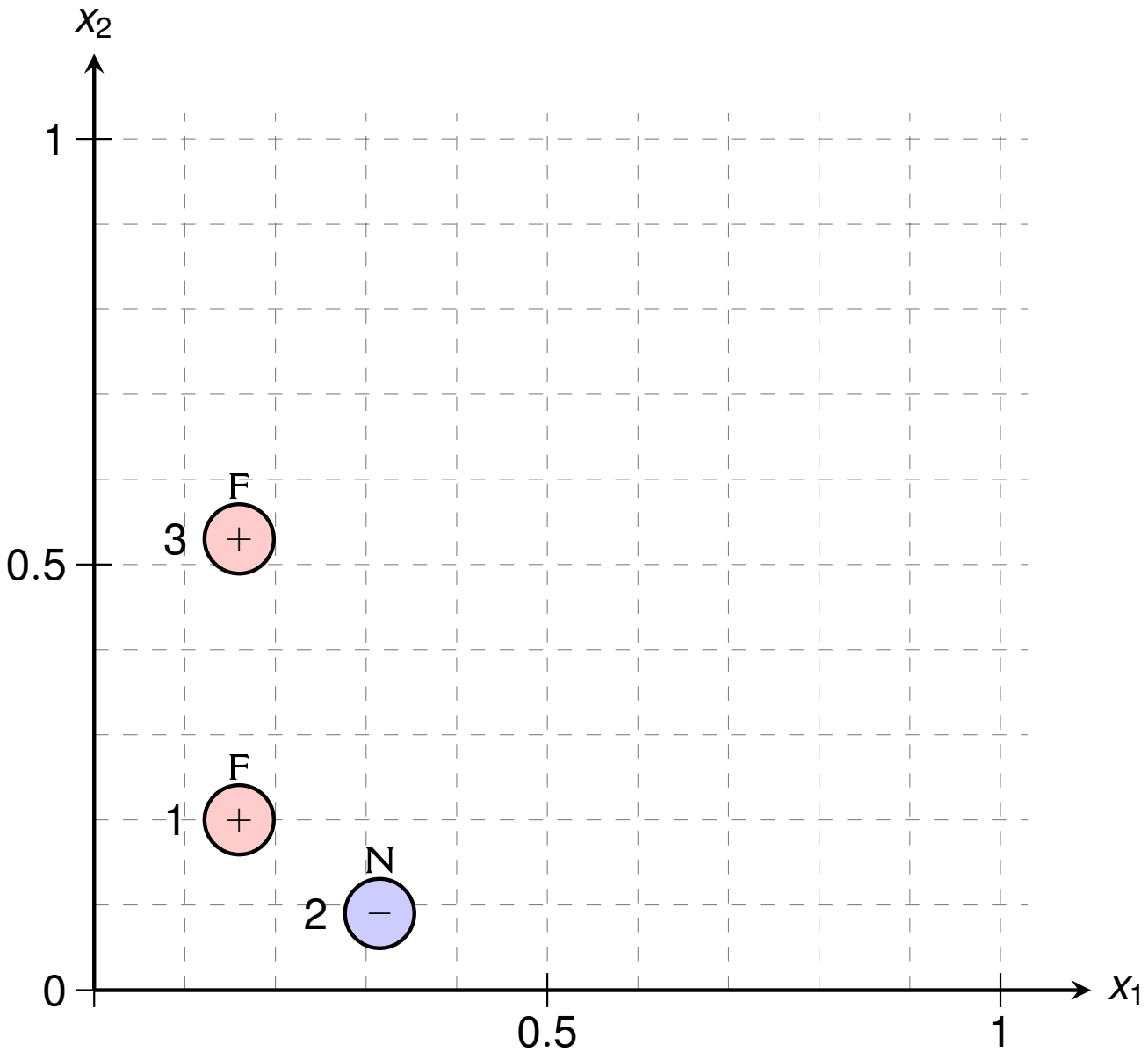
Schematic Example (3/3)



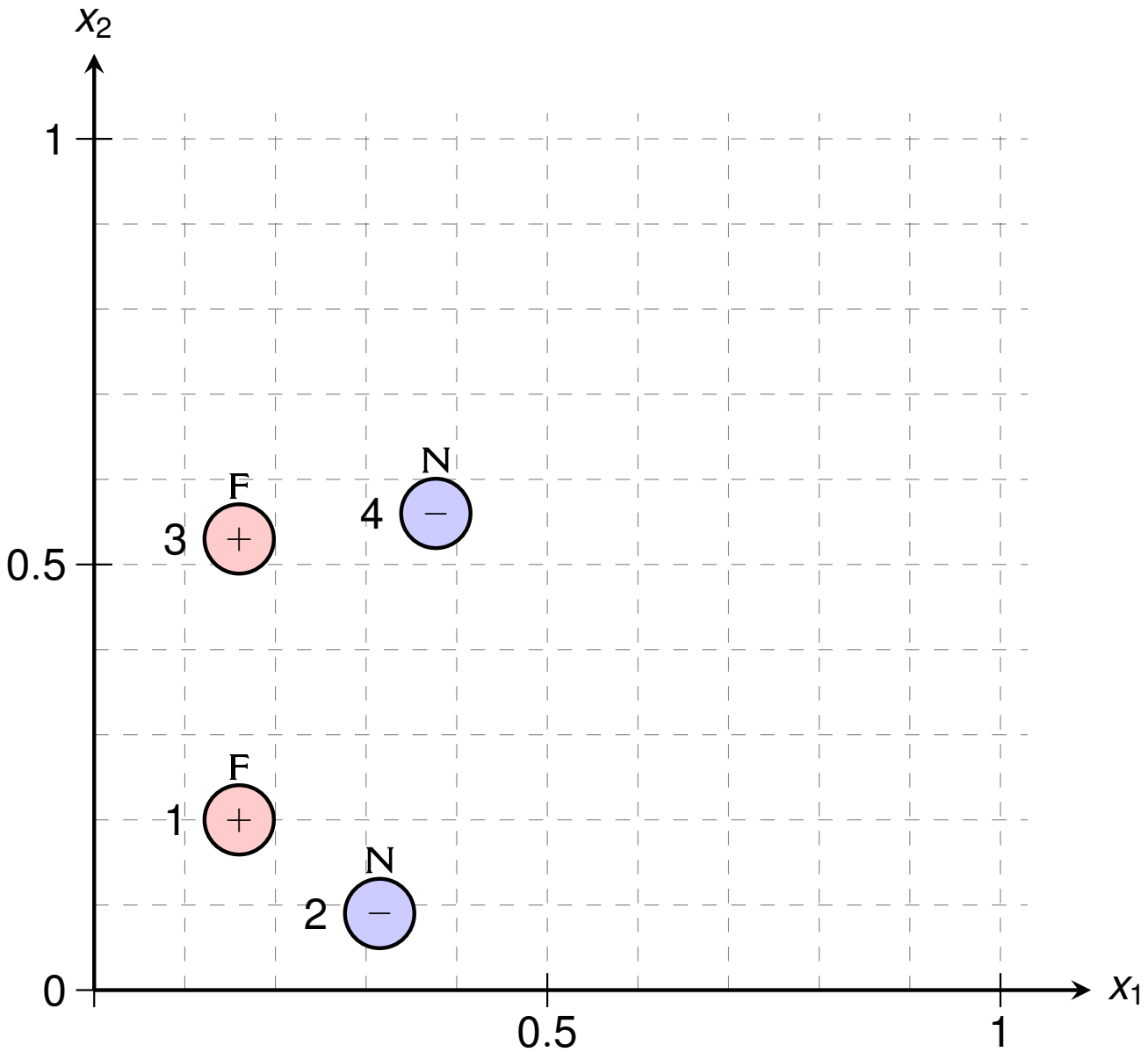
Schematic Example (3/3)



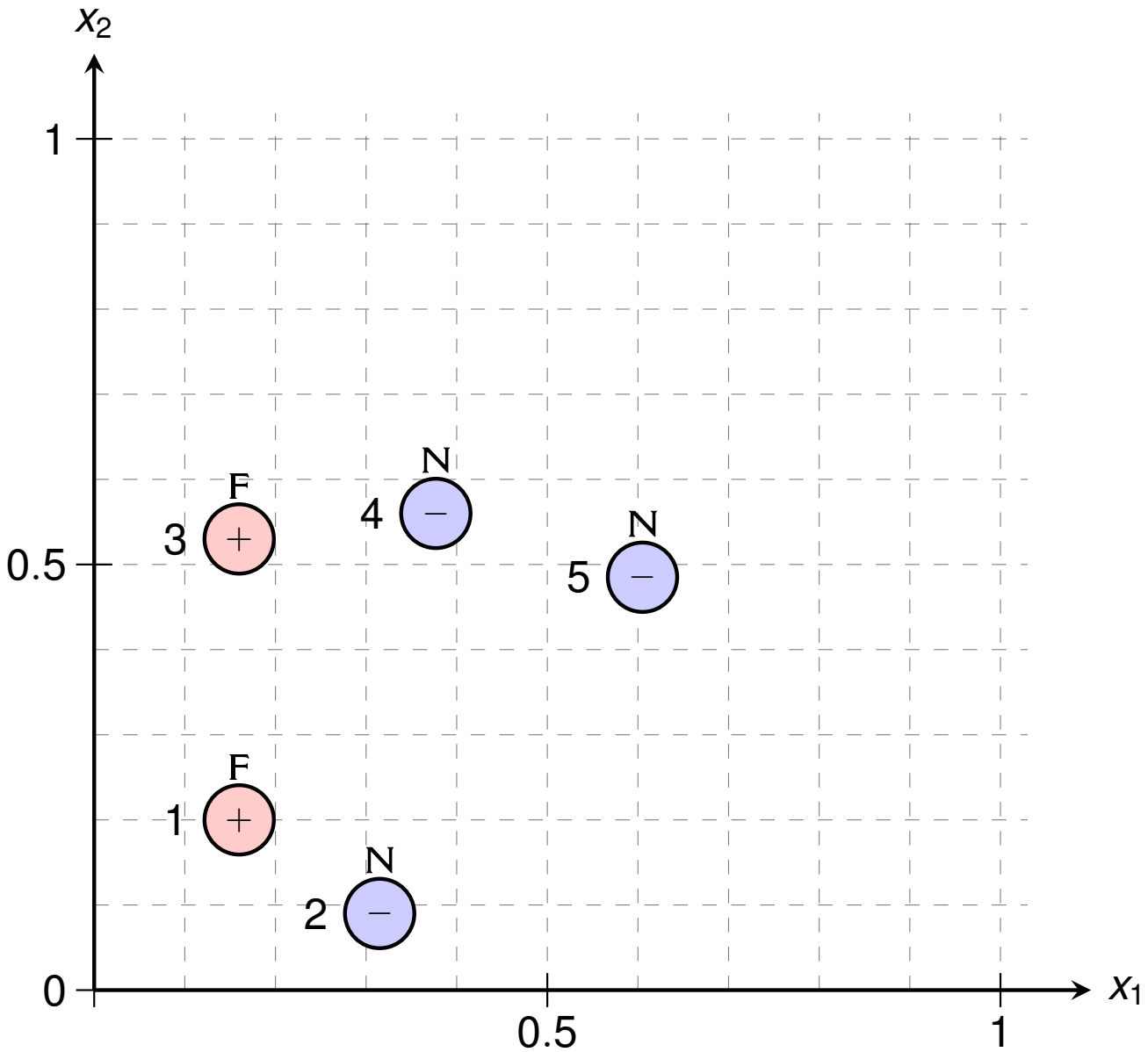
Schematic Example (3/3)



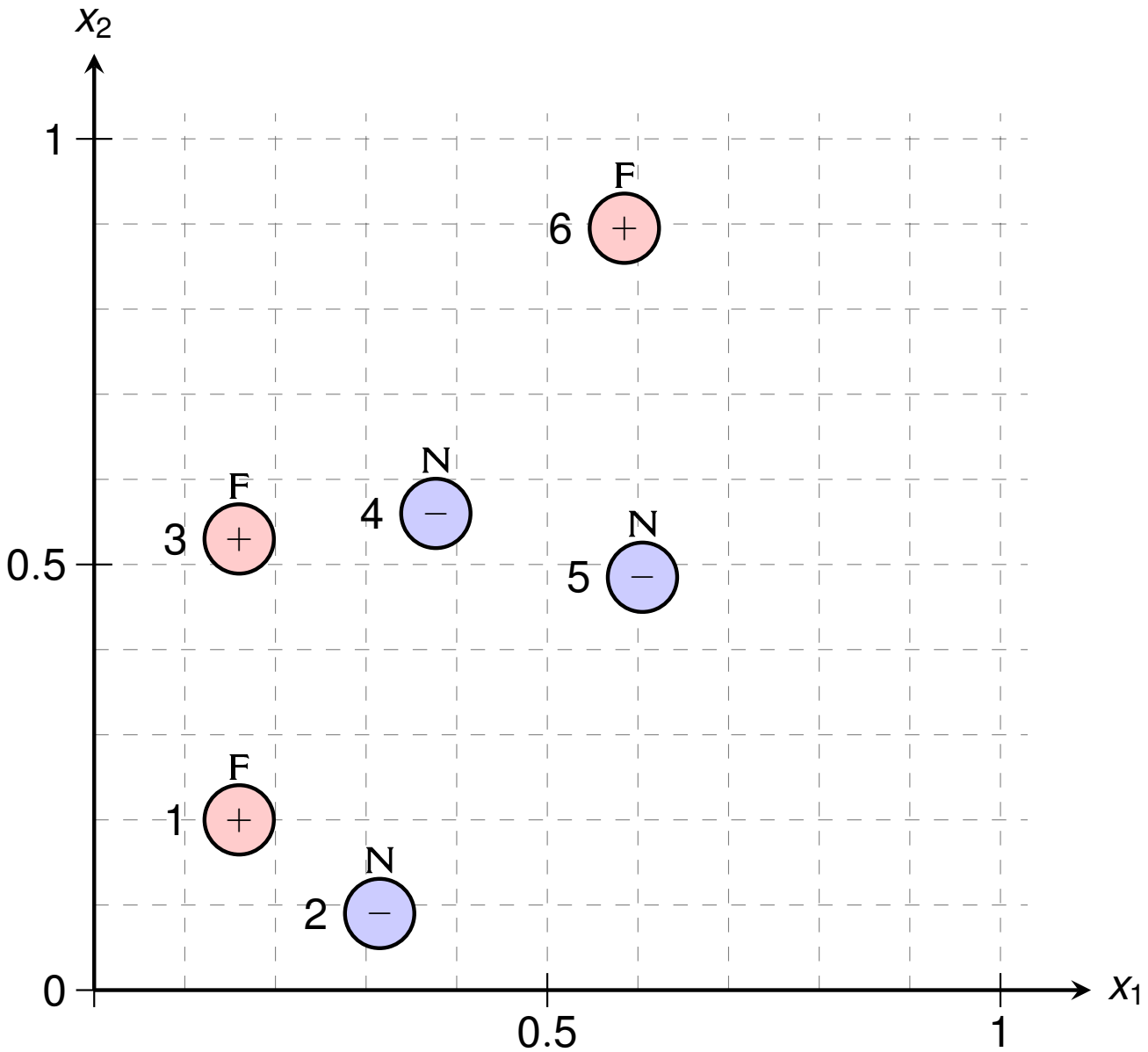
Schematic Example (3/3)



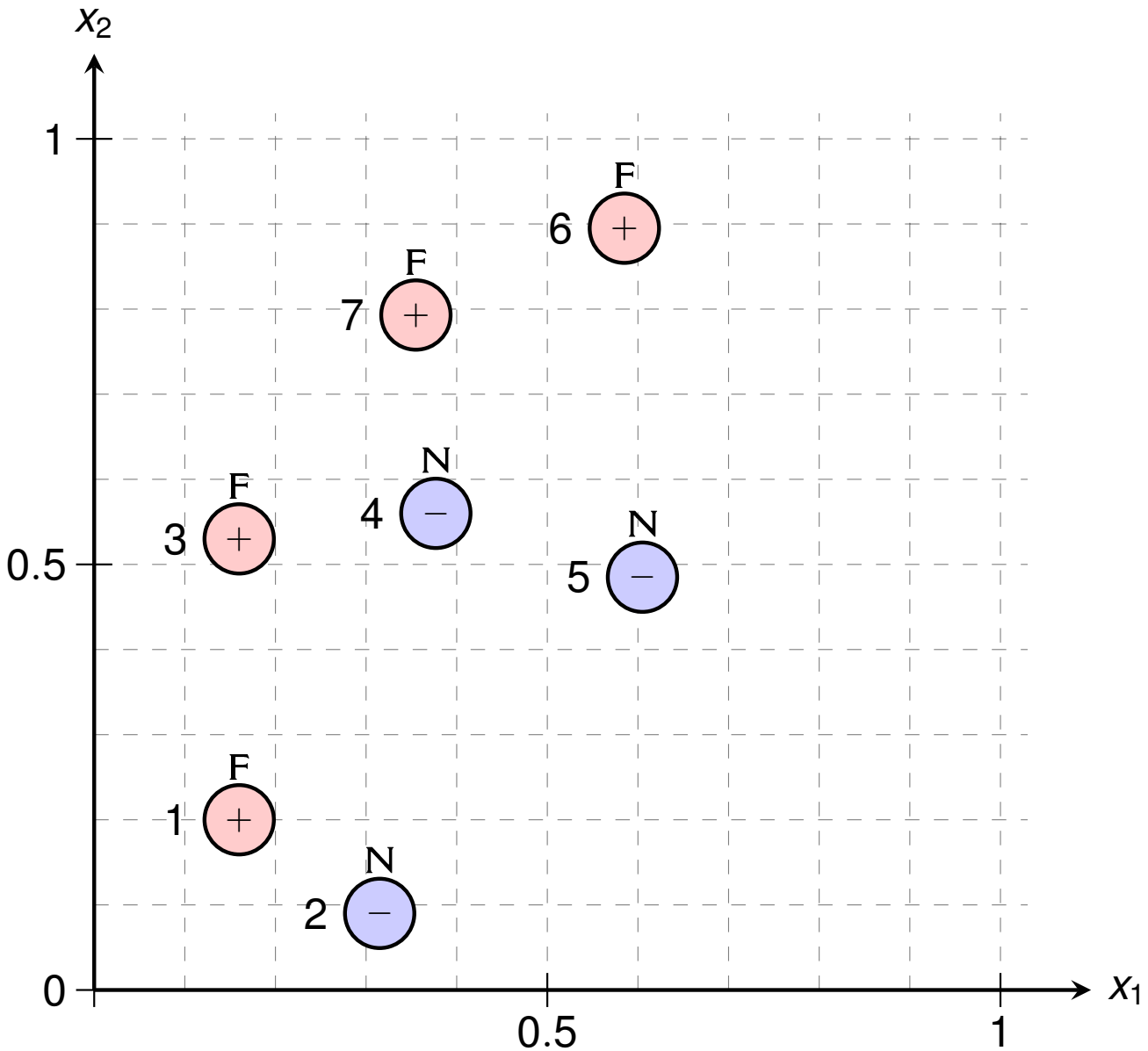
Schematic Example (3/3)



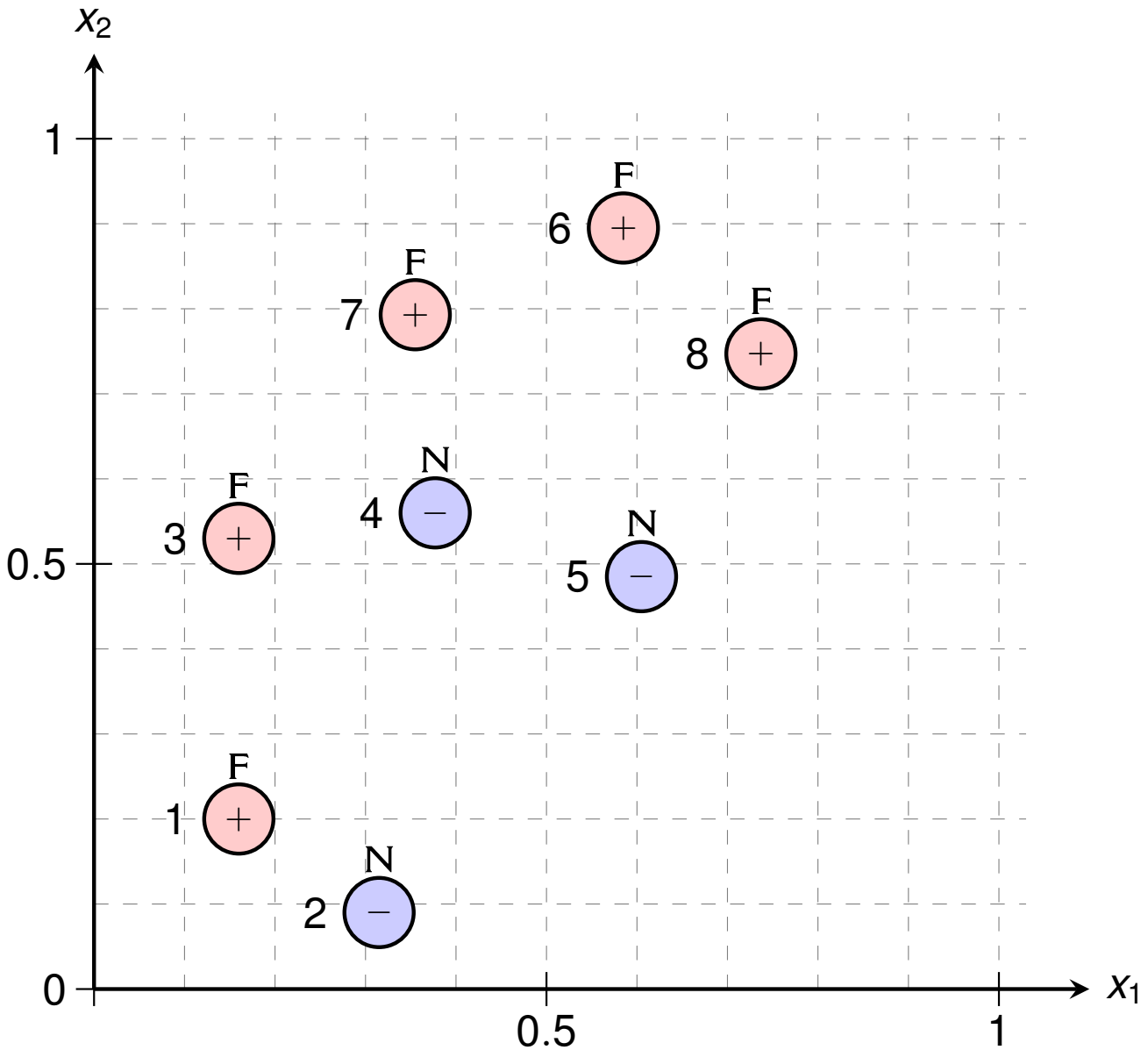
Schematic Example (3/3)



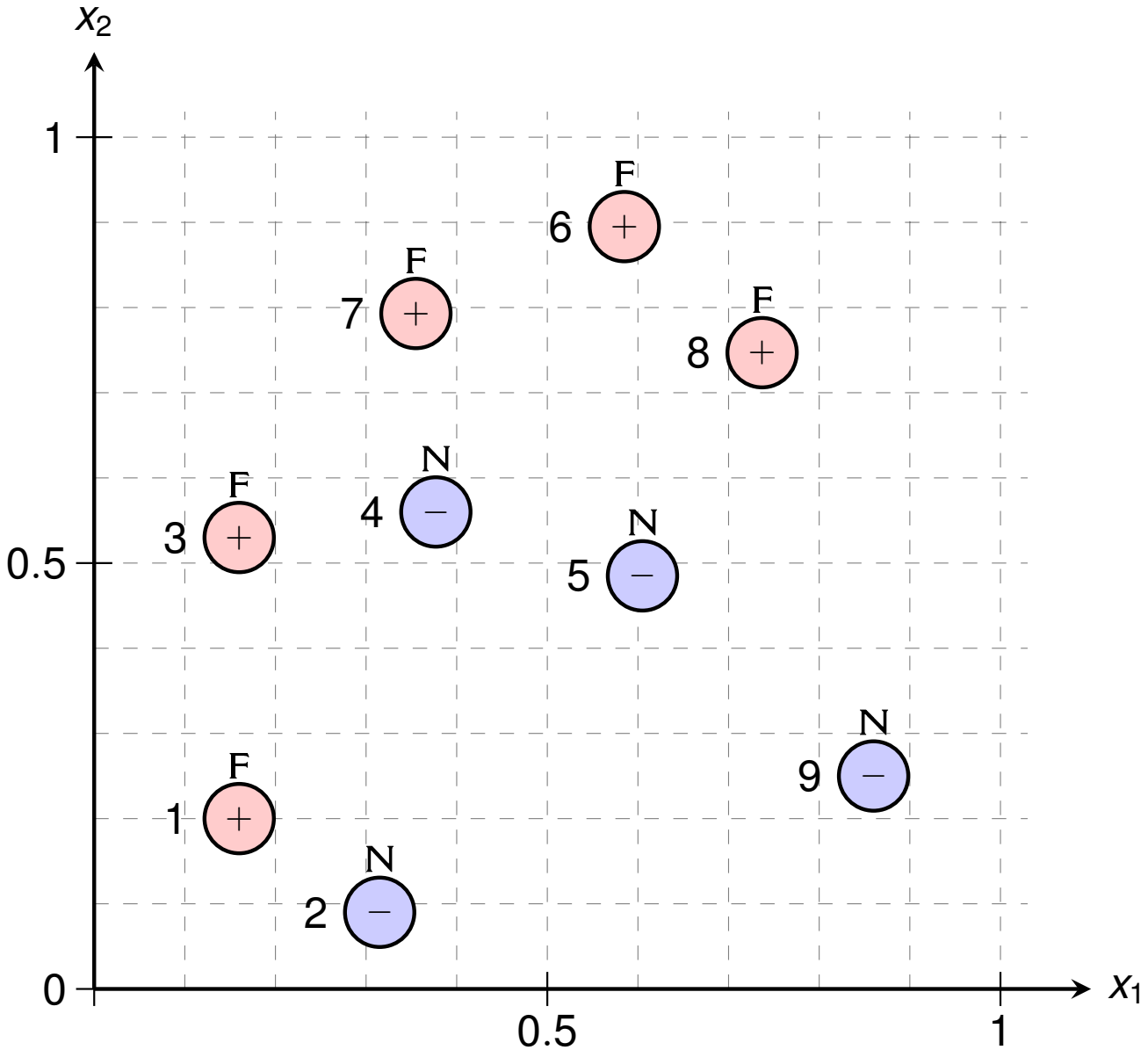
Schematic Example (3/3)



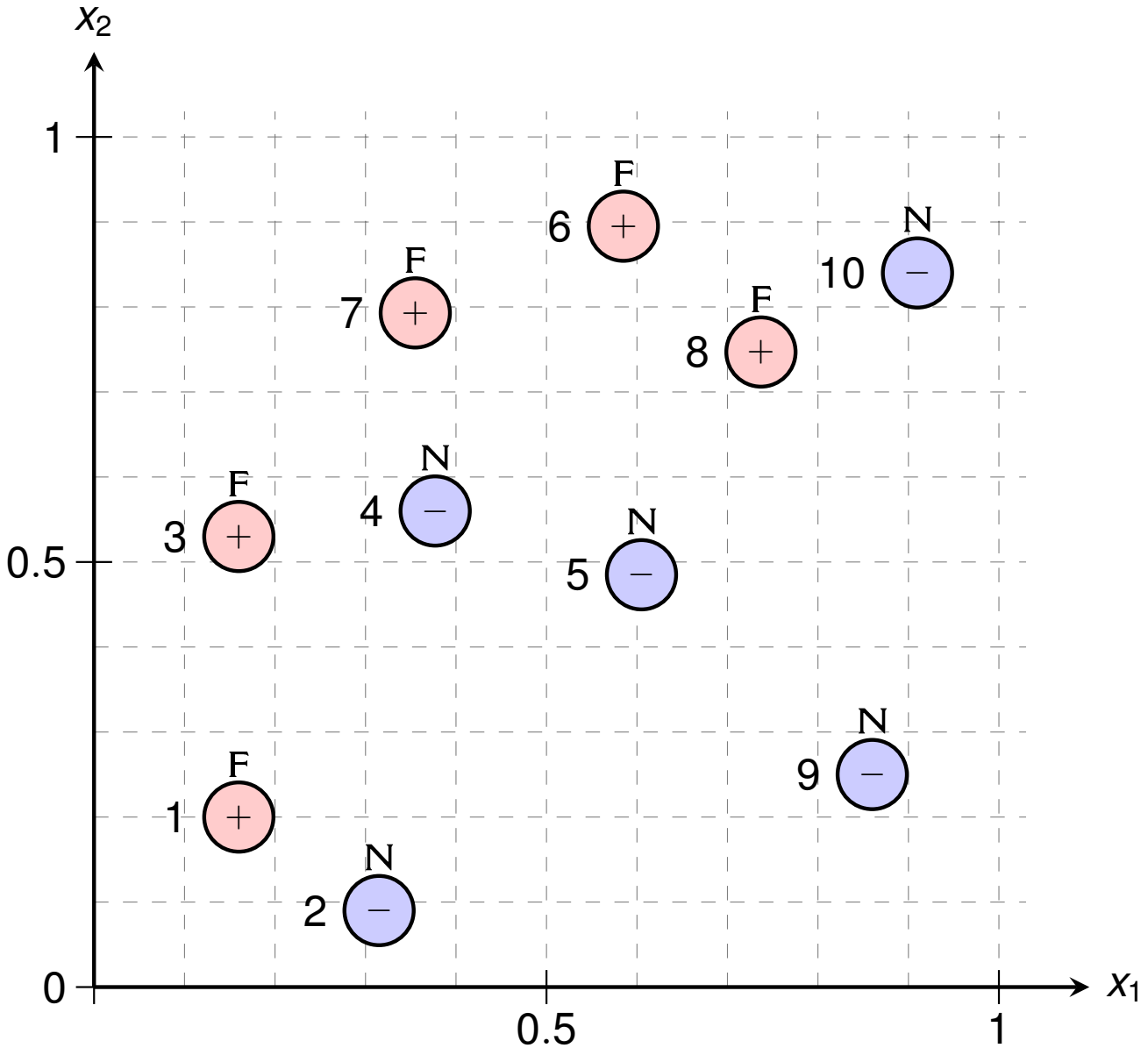
Schematic Example (3/3)



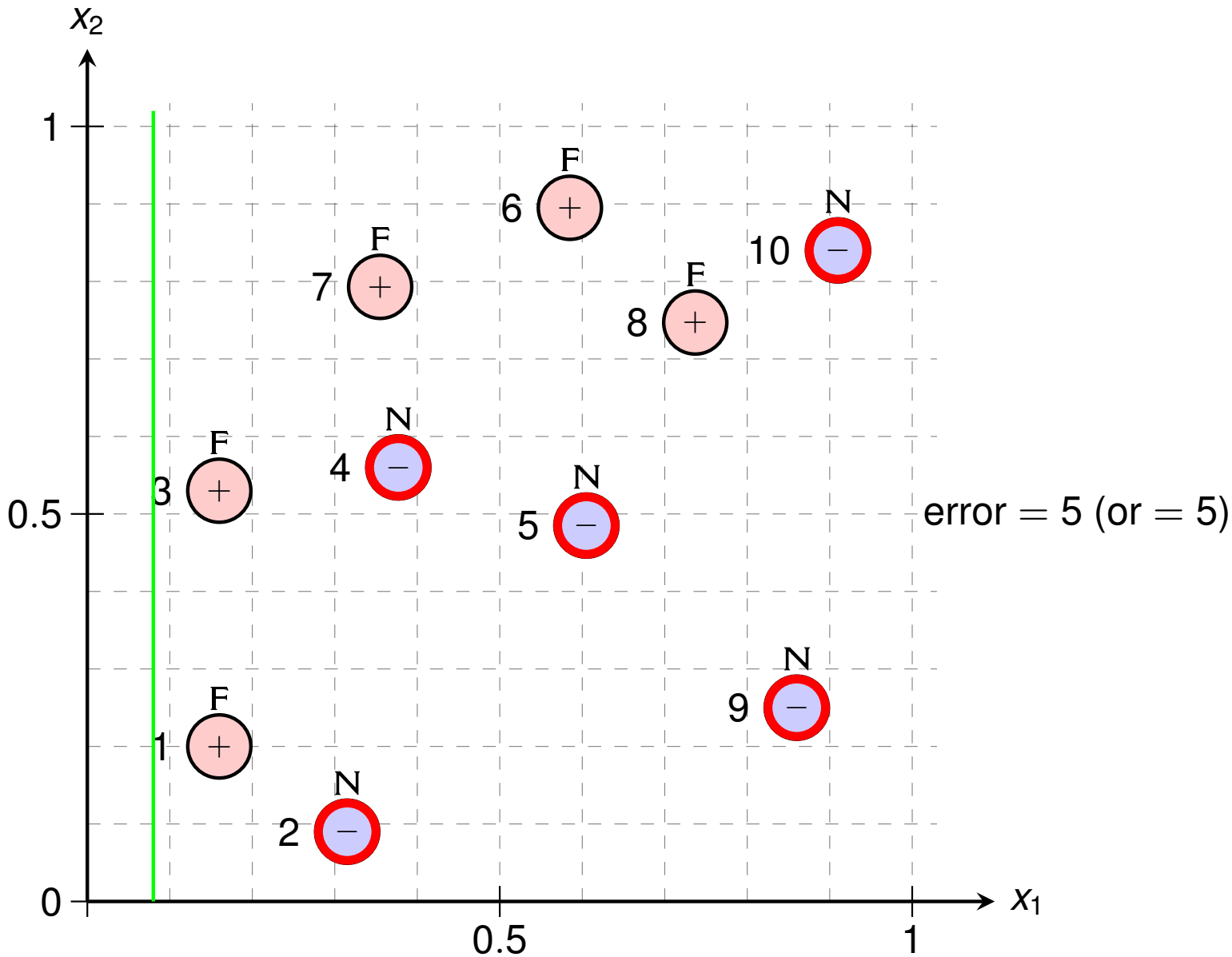
Schematic Example (3/3)



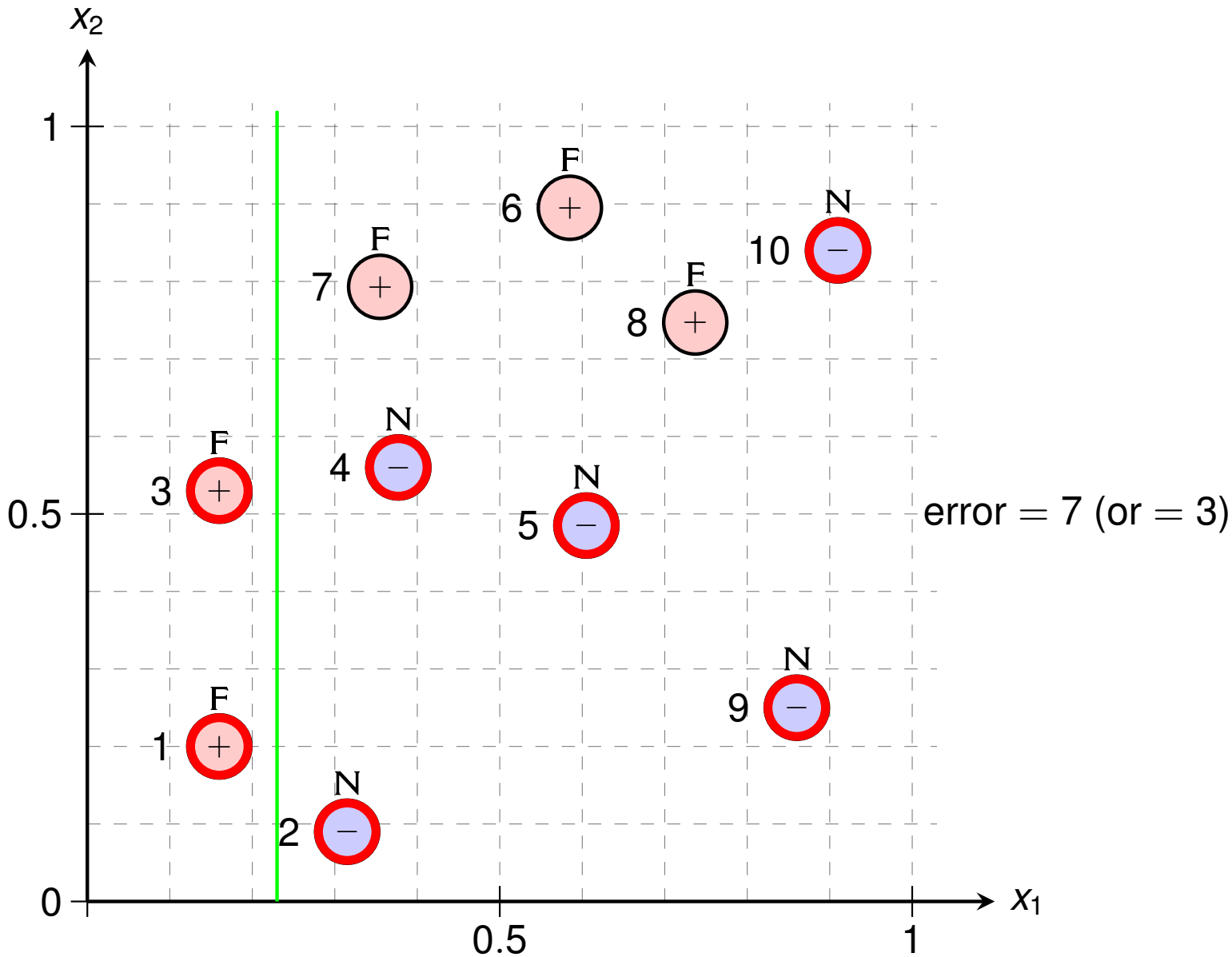
Schematic Example (3/3)



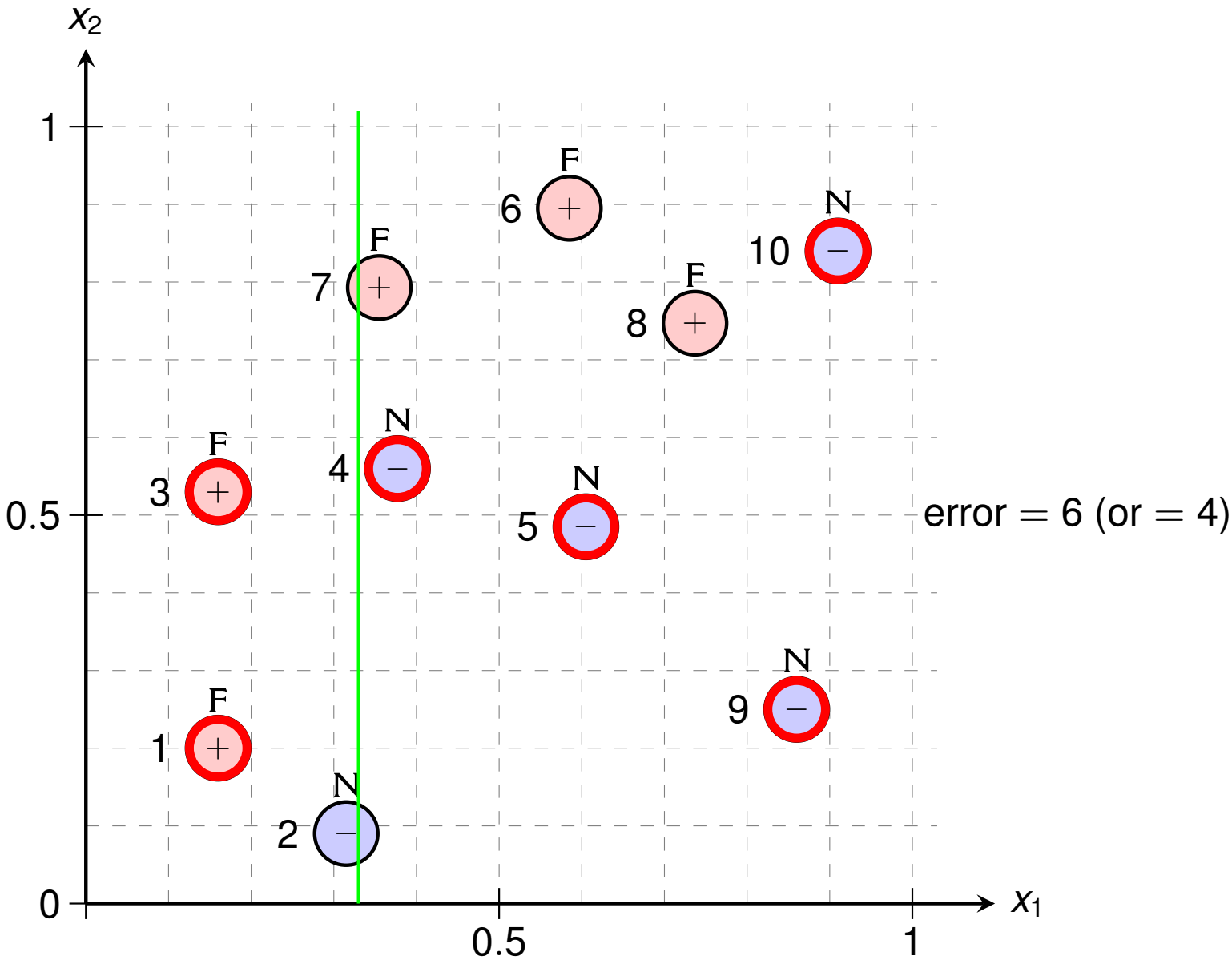
Schematic Example (3/3)



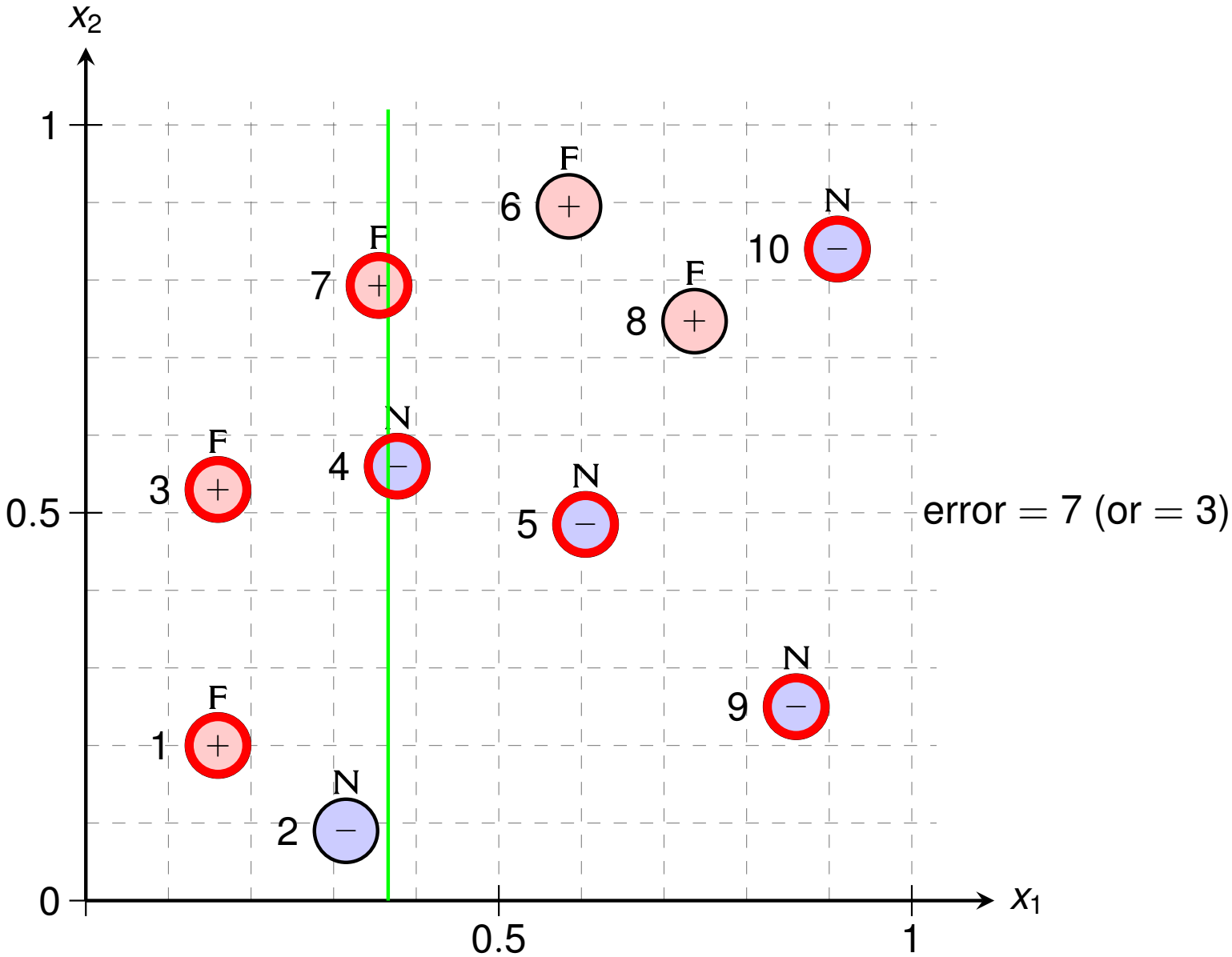
Schematic Example (3/3)



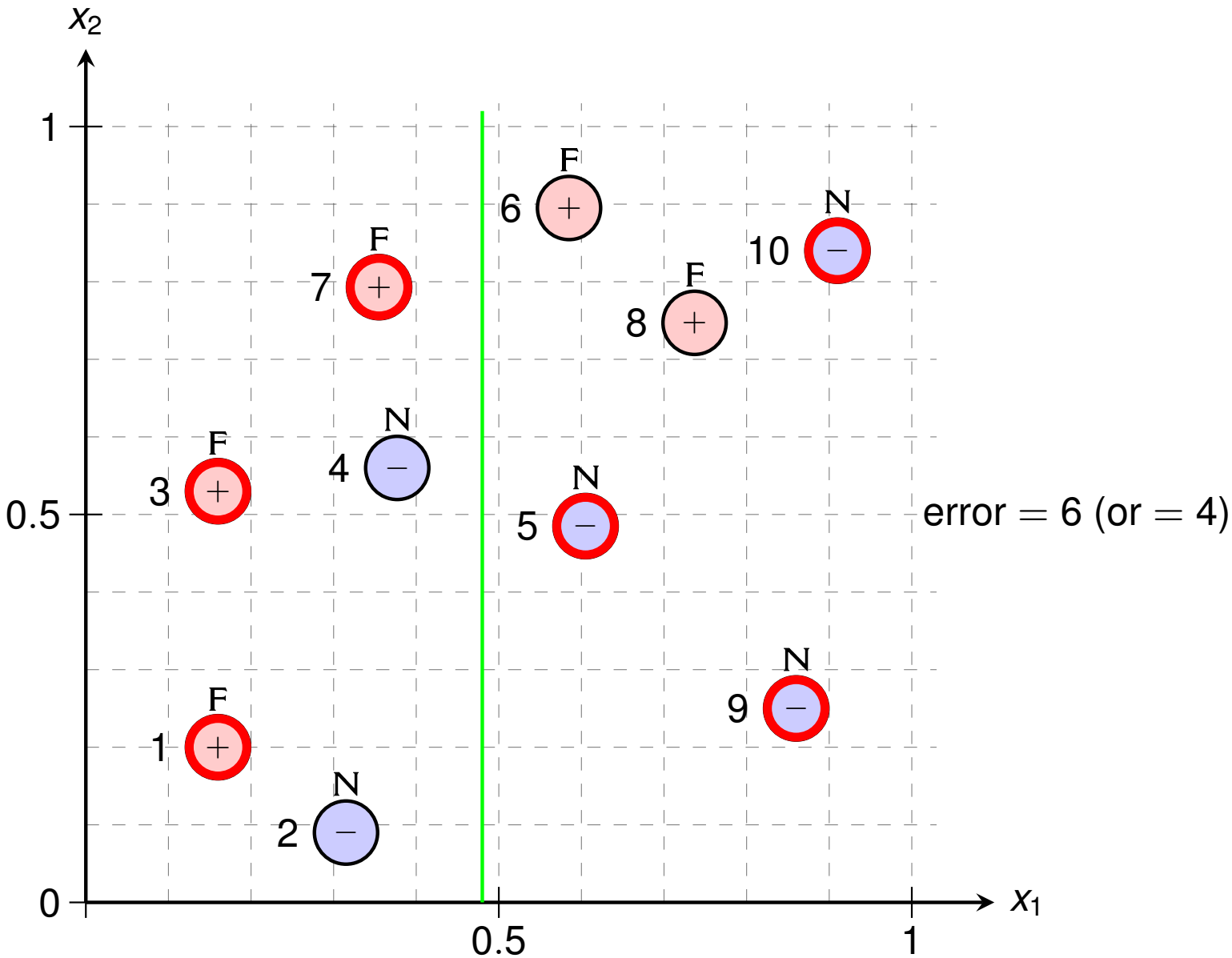
Schematic Example (3/3)



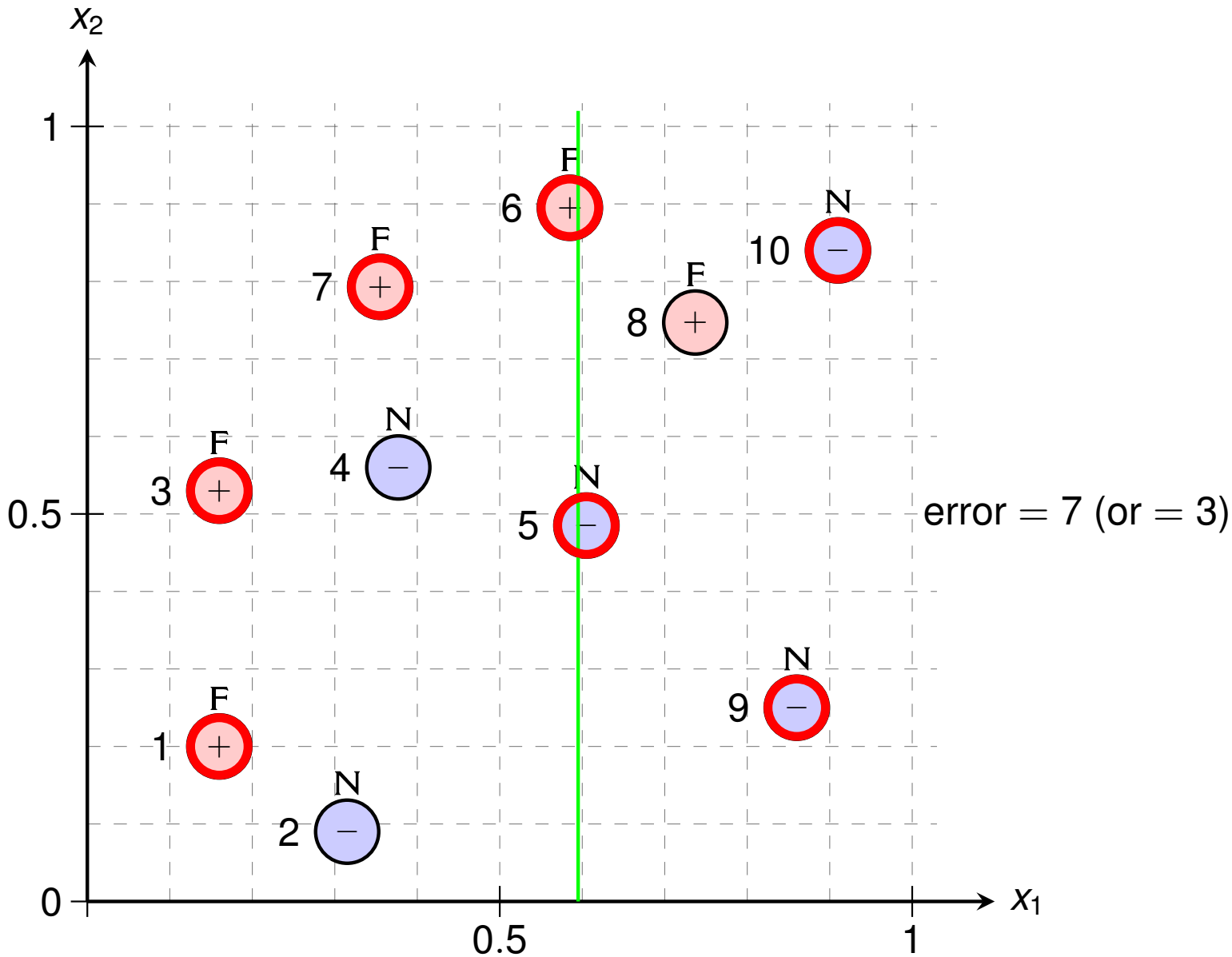
Schematic Example (3/3)



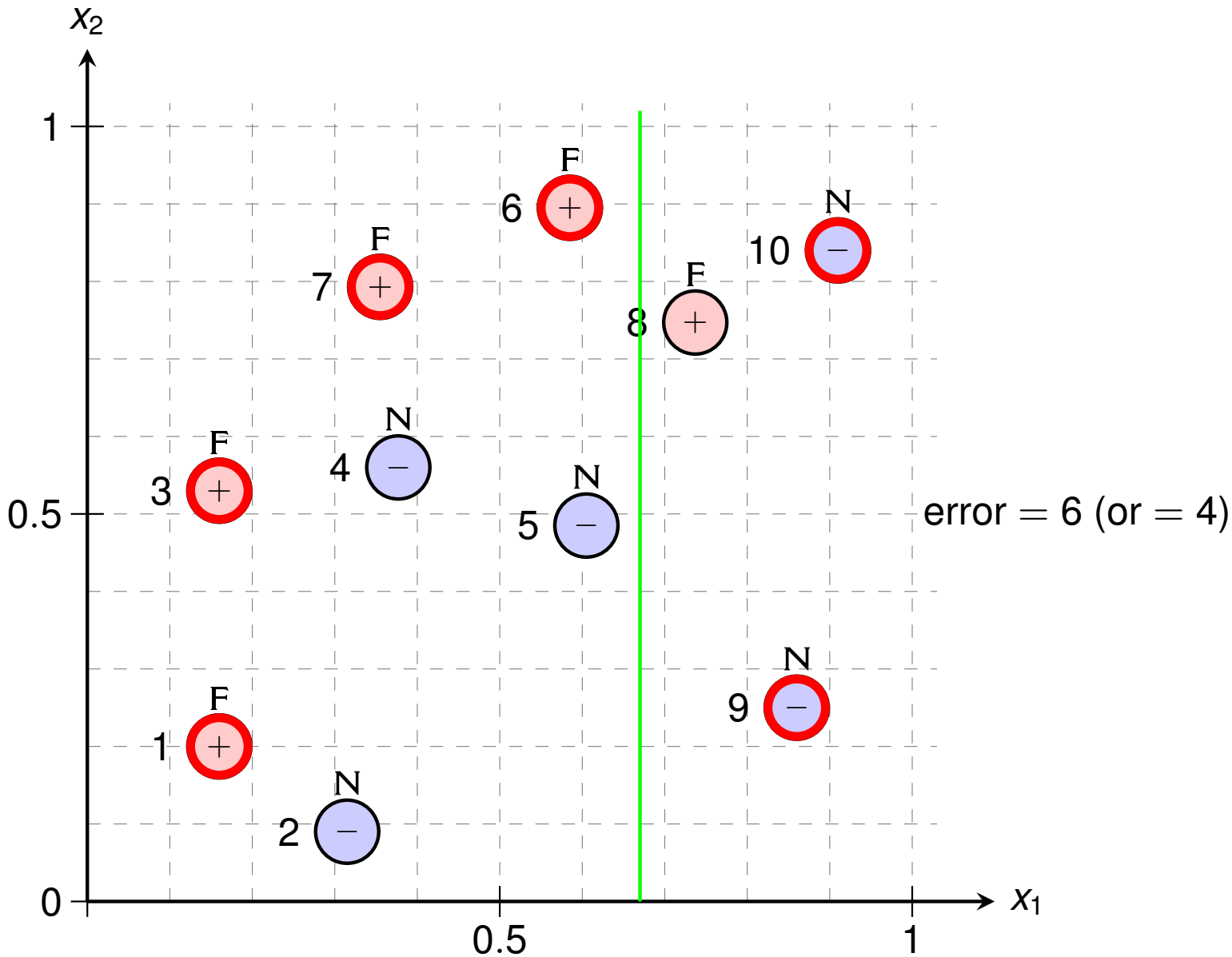
Schematic Example (3/3)



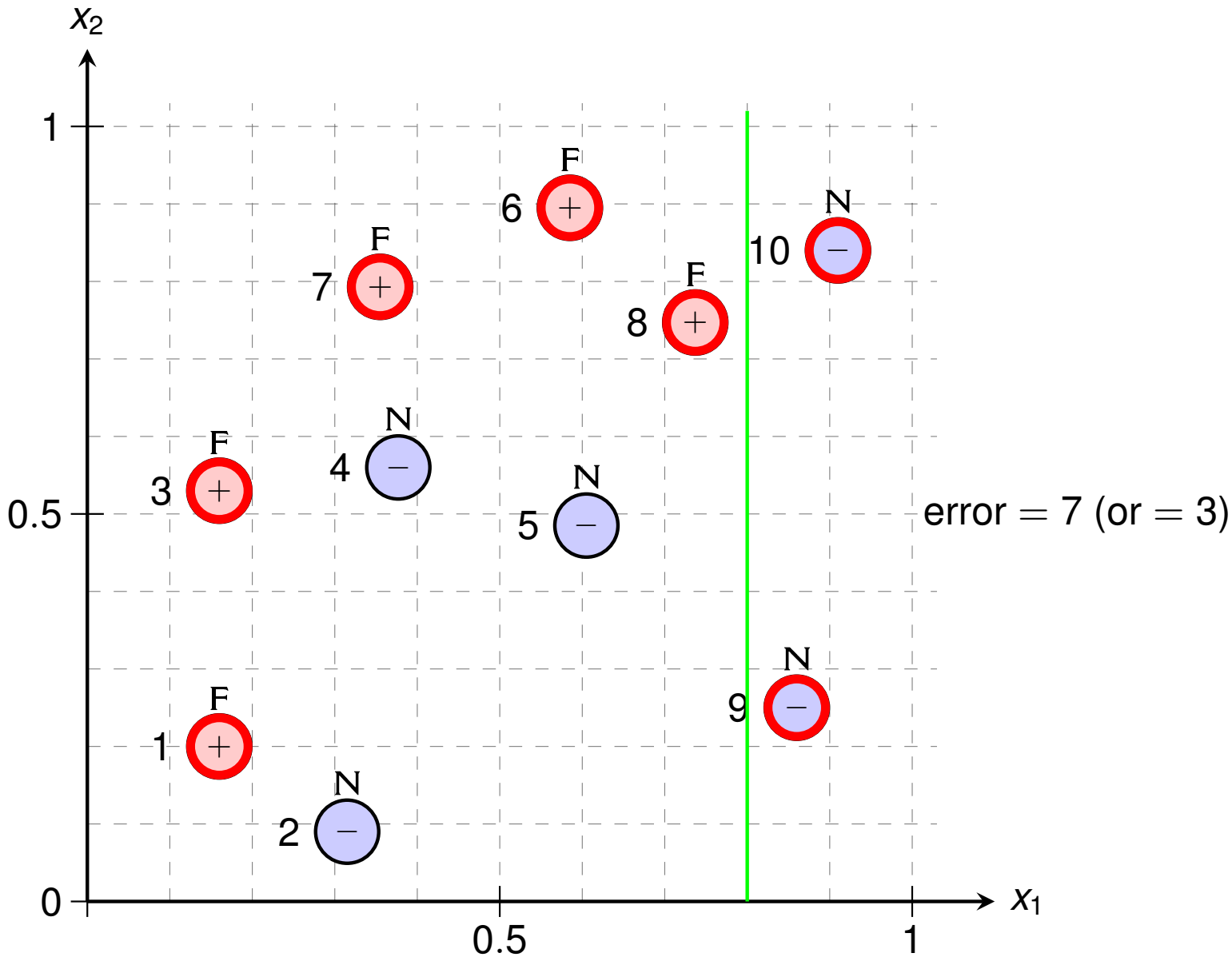
Schematic Example (3/3)



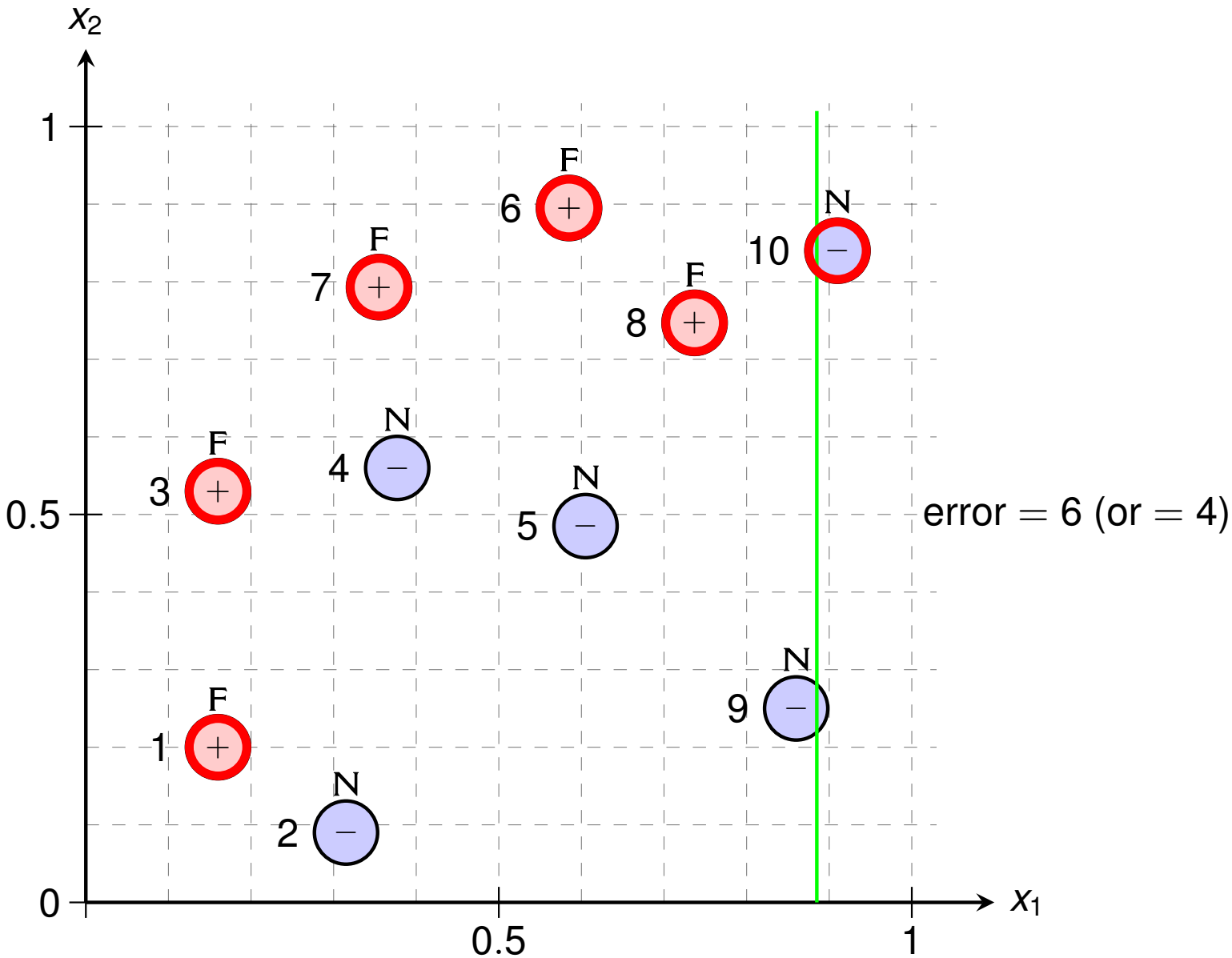
Schematic Example (3/3)



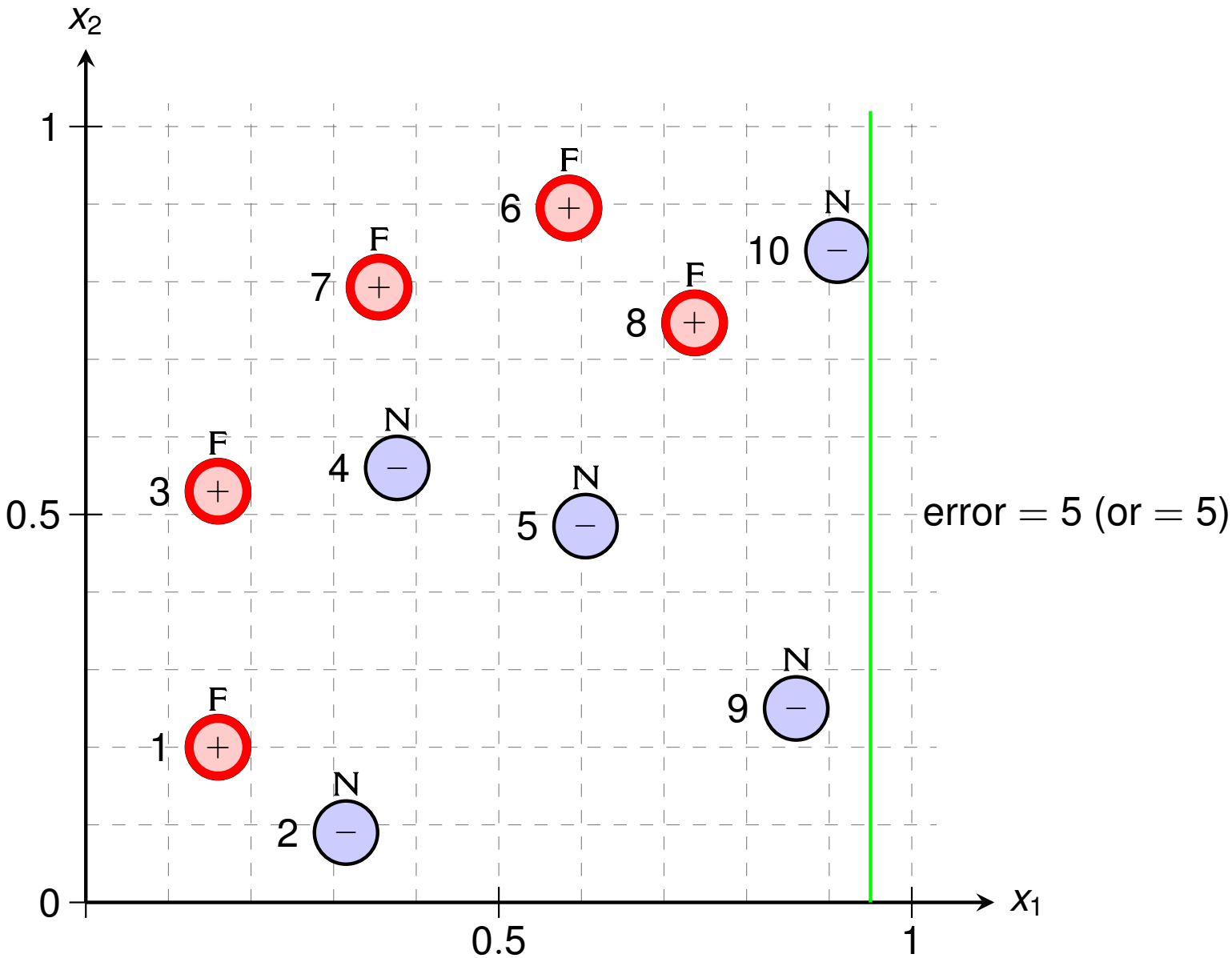
Schematic Example (3/3)



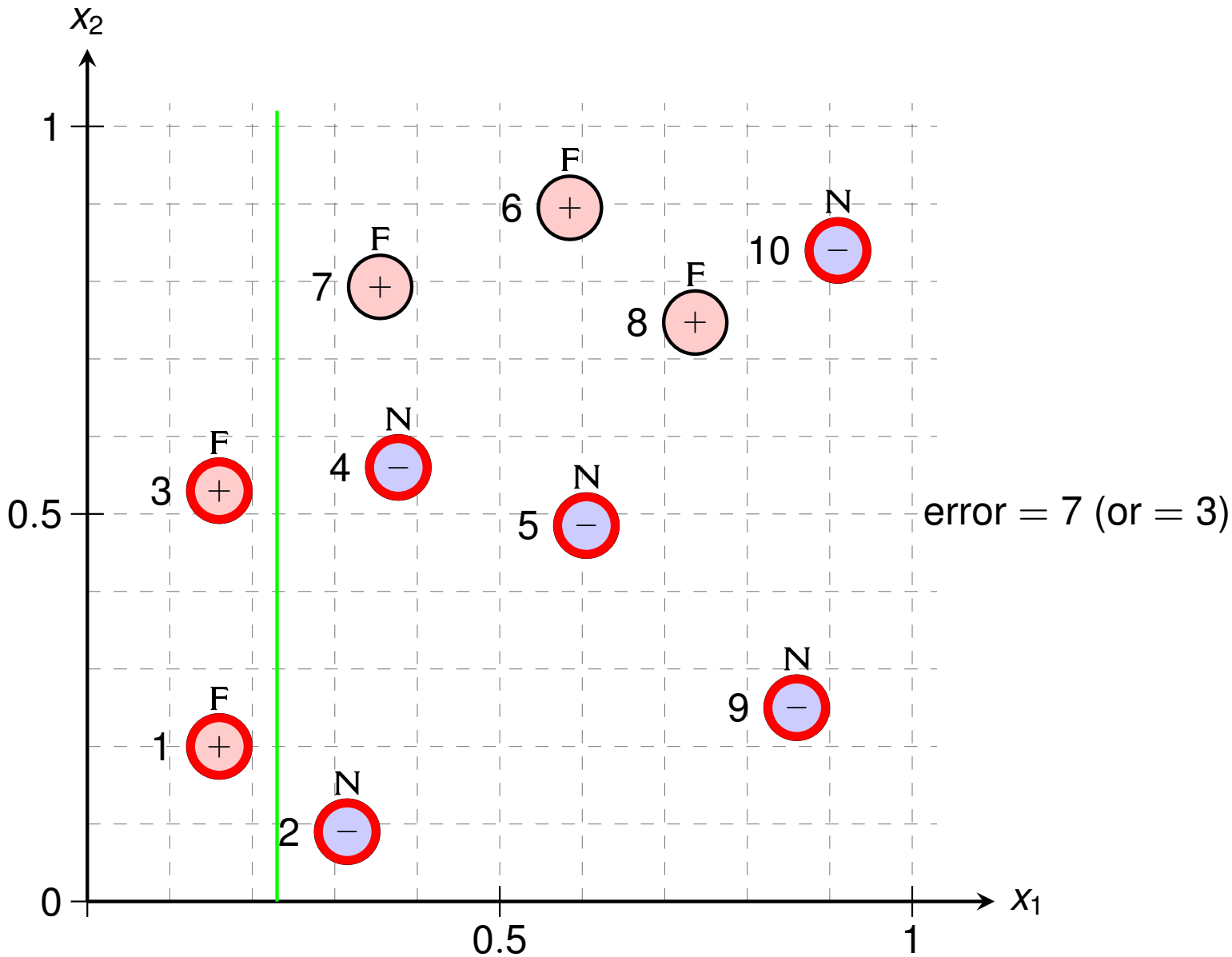
Schematic Example (3/3)



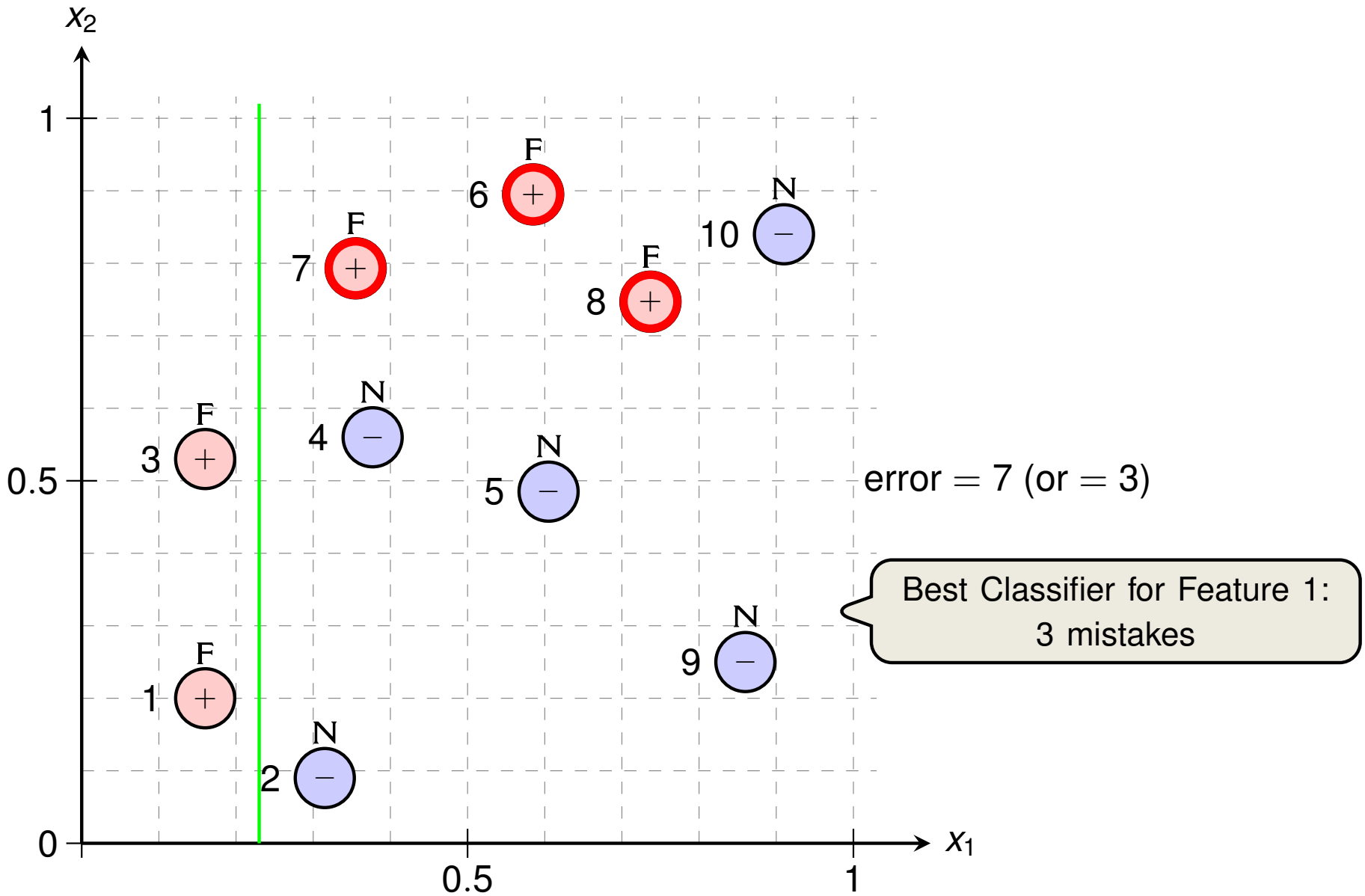
Schematic Example (3/3)



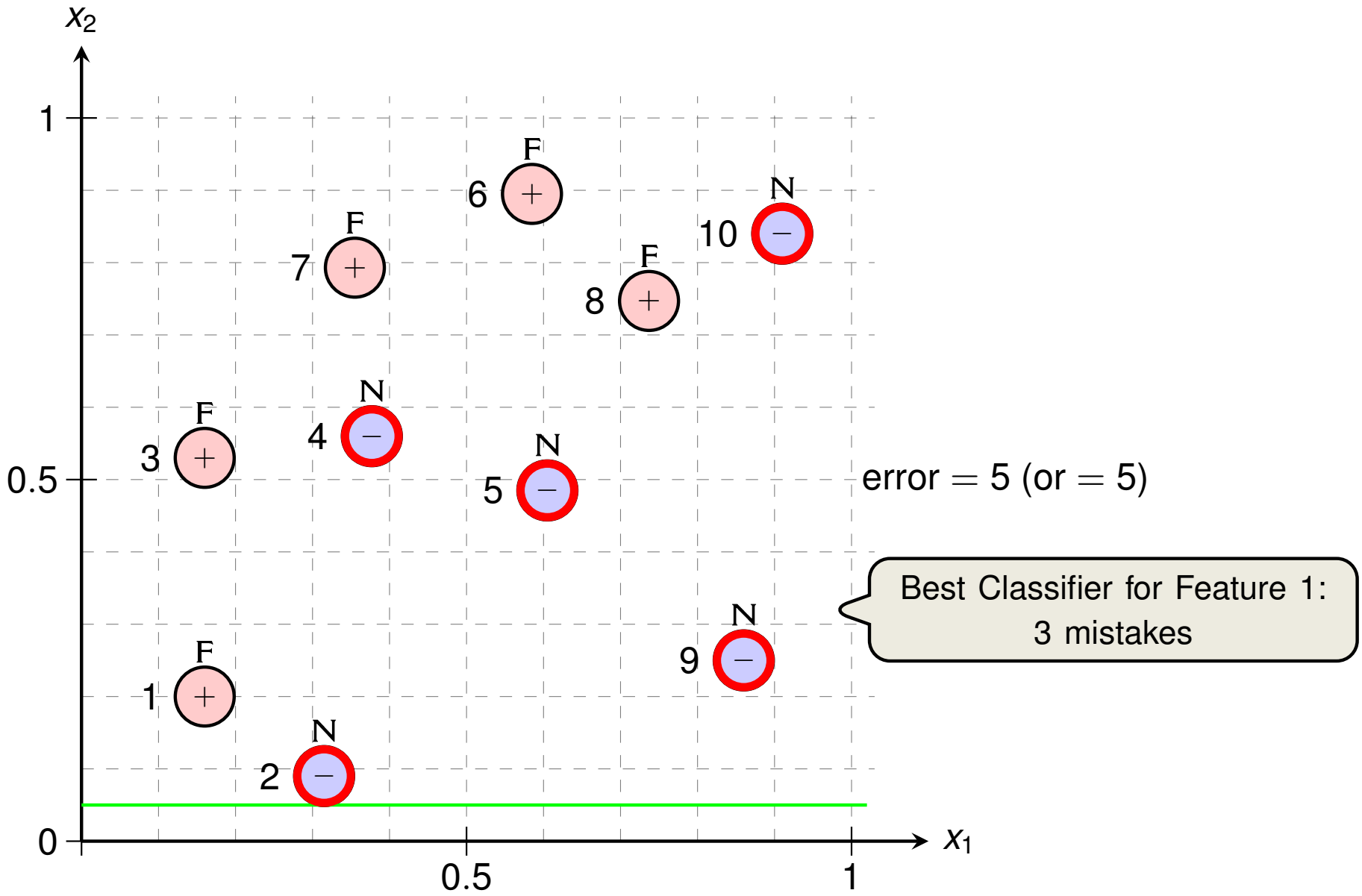
Schematic Example (3/3)



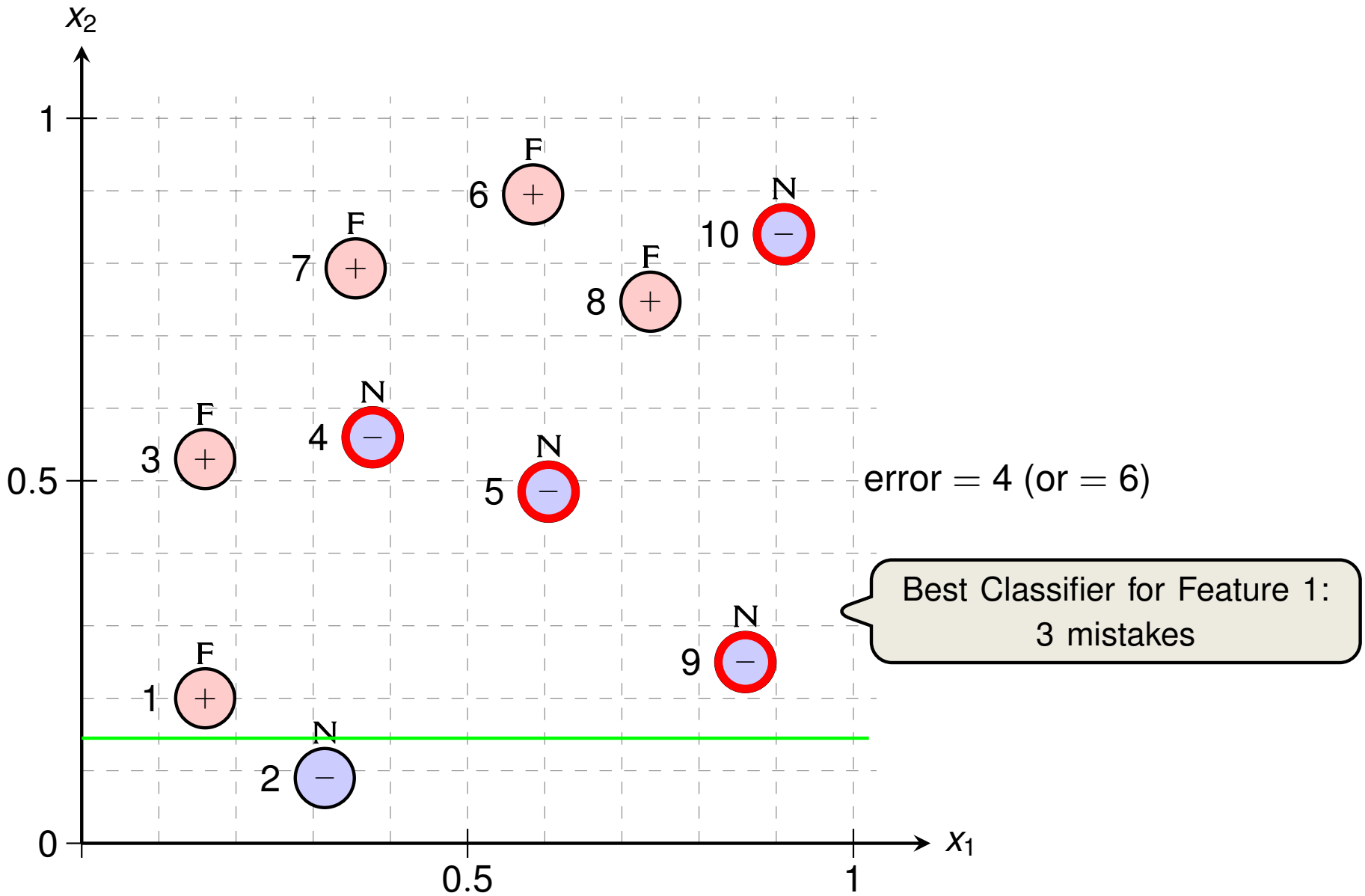
Schematic Example (3/3)



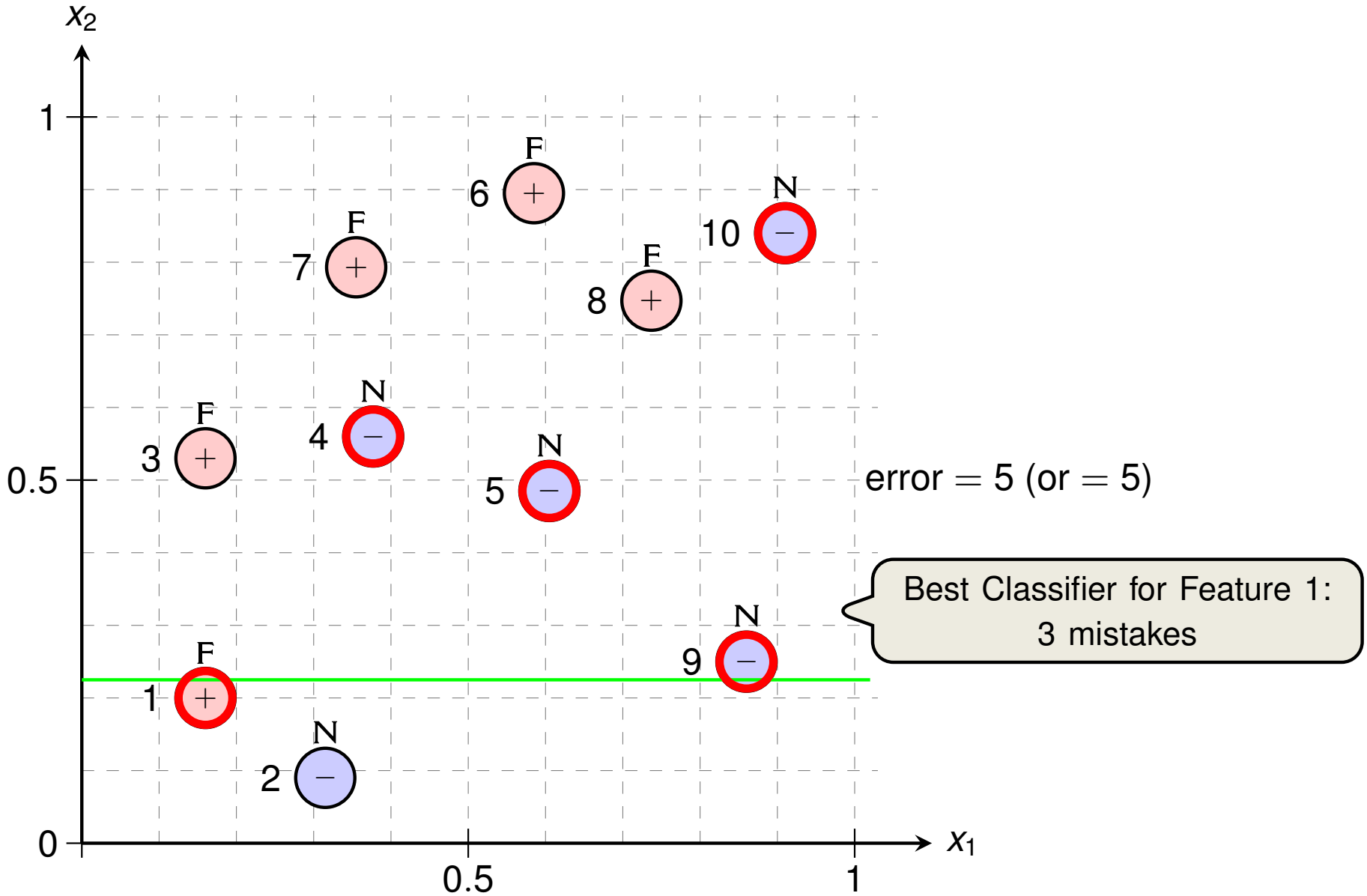
Schematic Example (3/3)



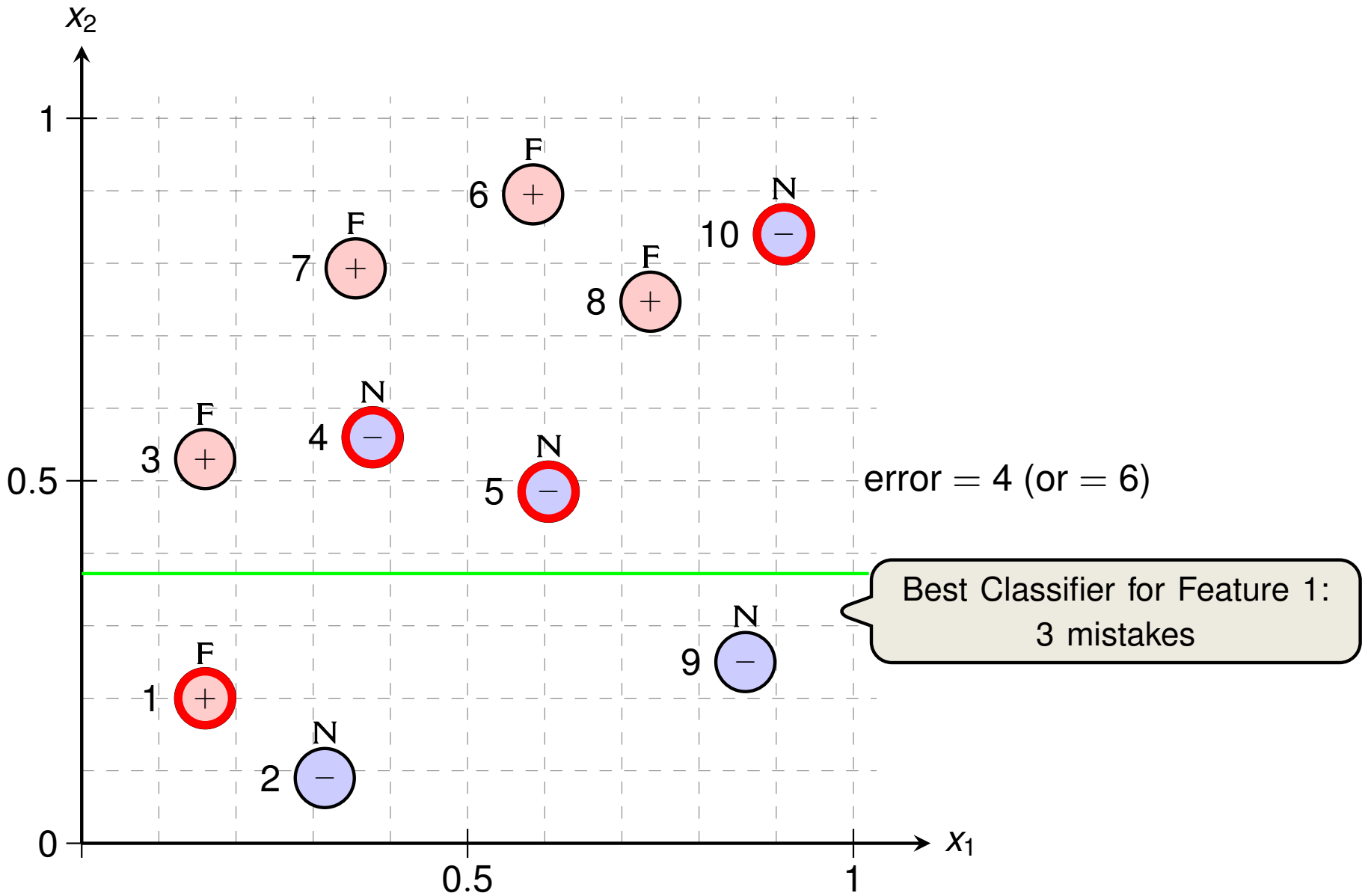
Schematic Example (3/3)



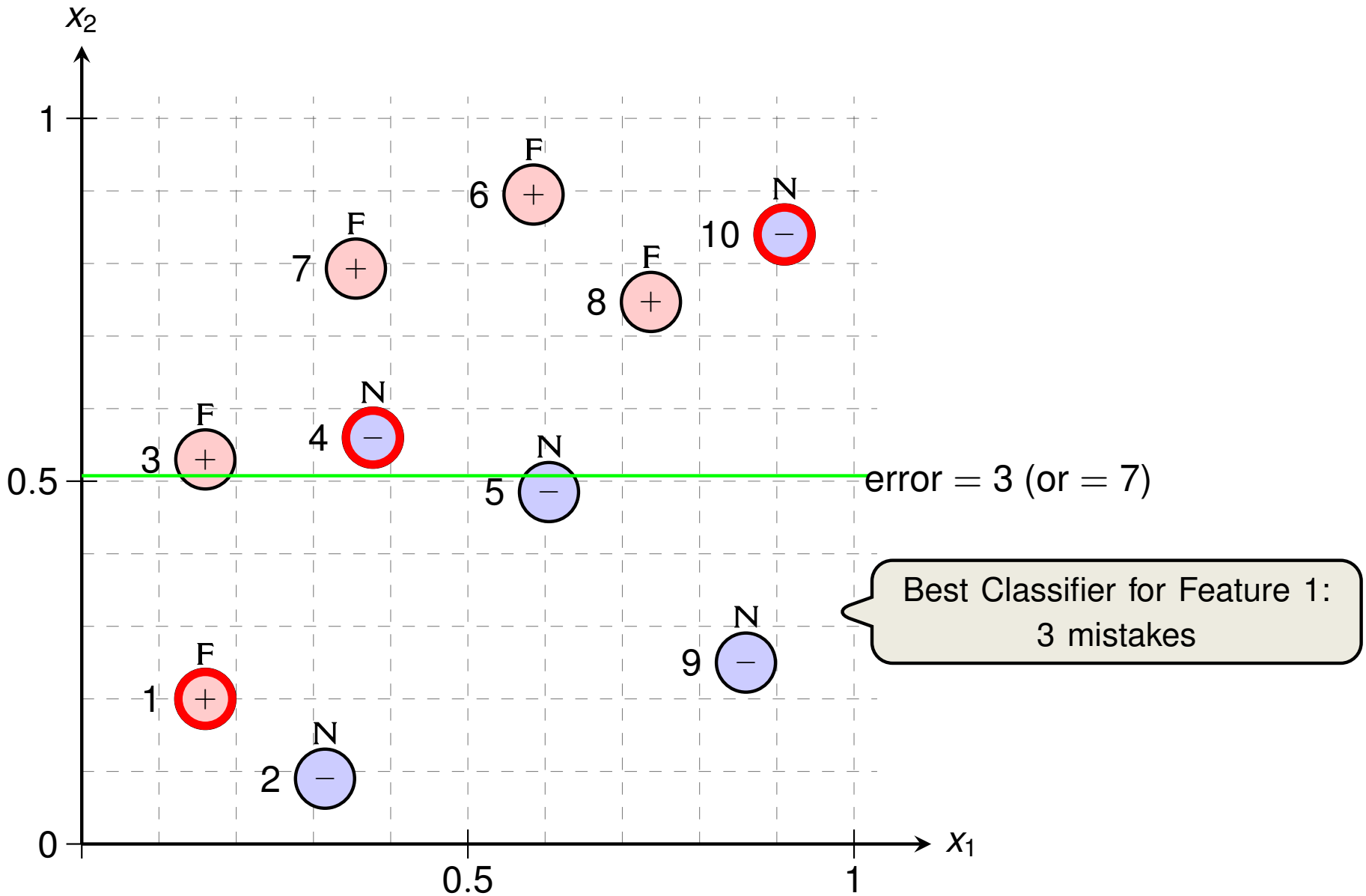
Schematic Example (3/3)



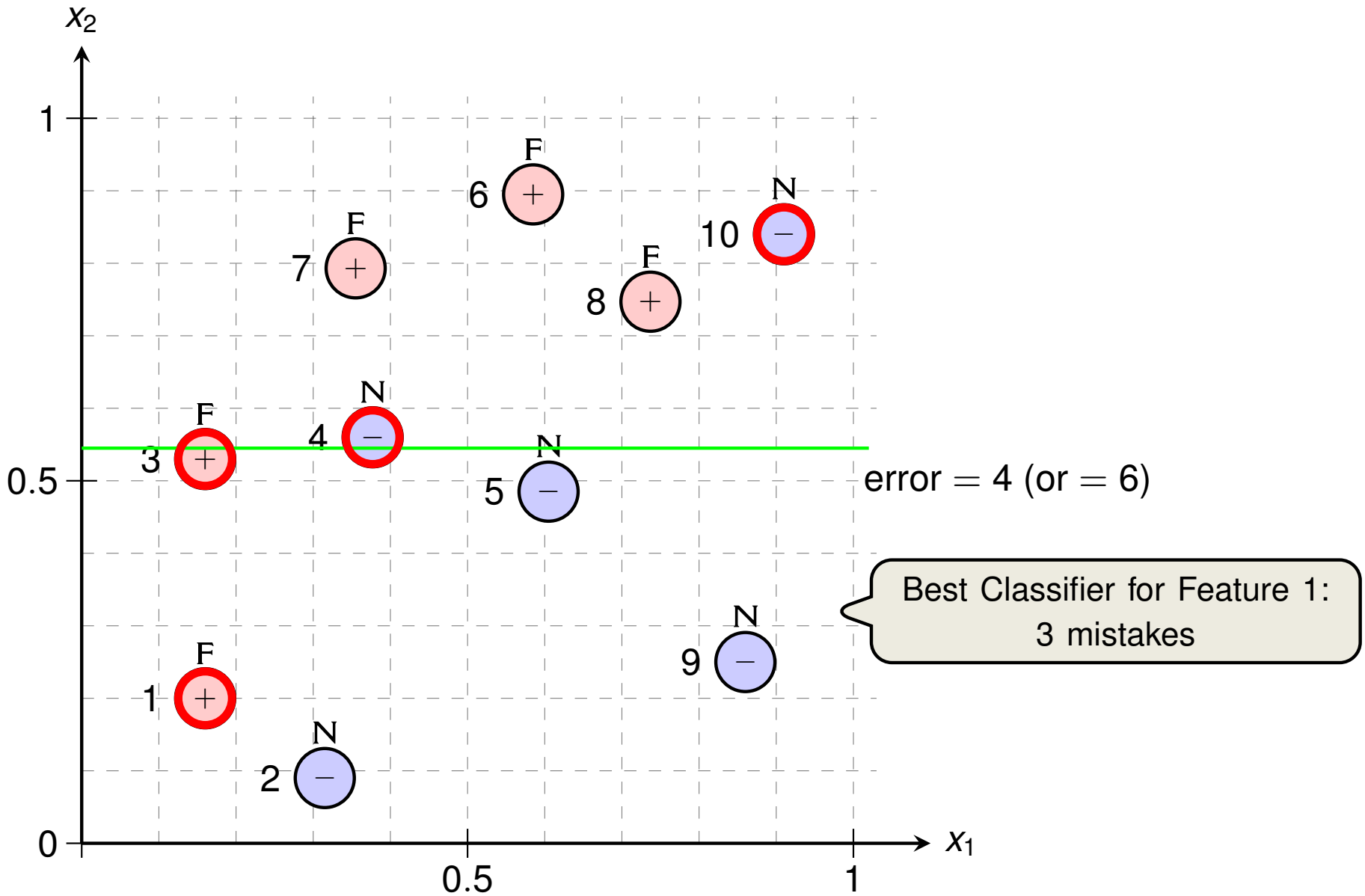
Schematic Example (3/3)



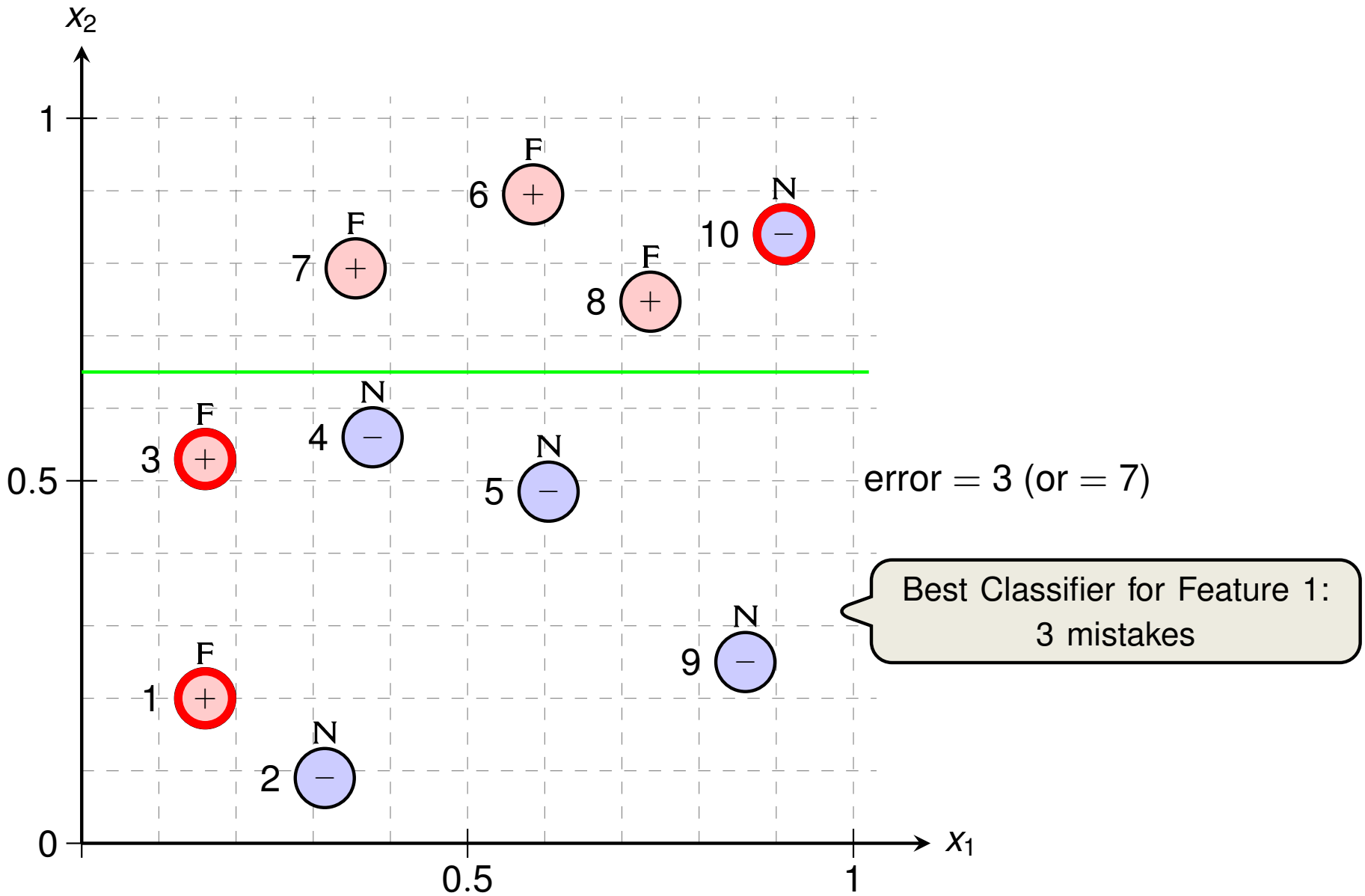
Schematic Example (3/3)



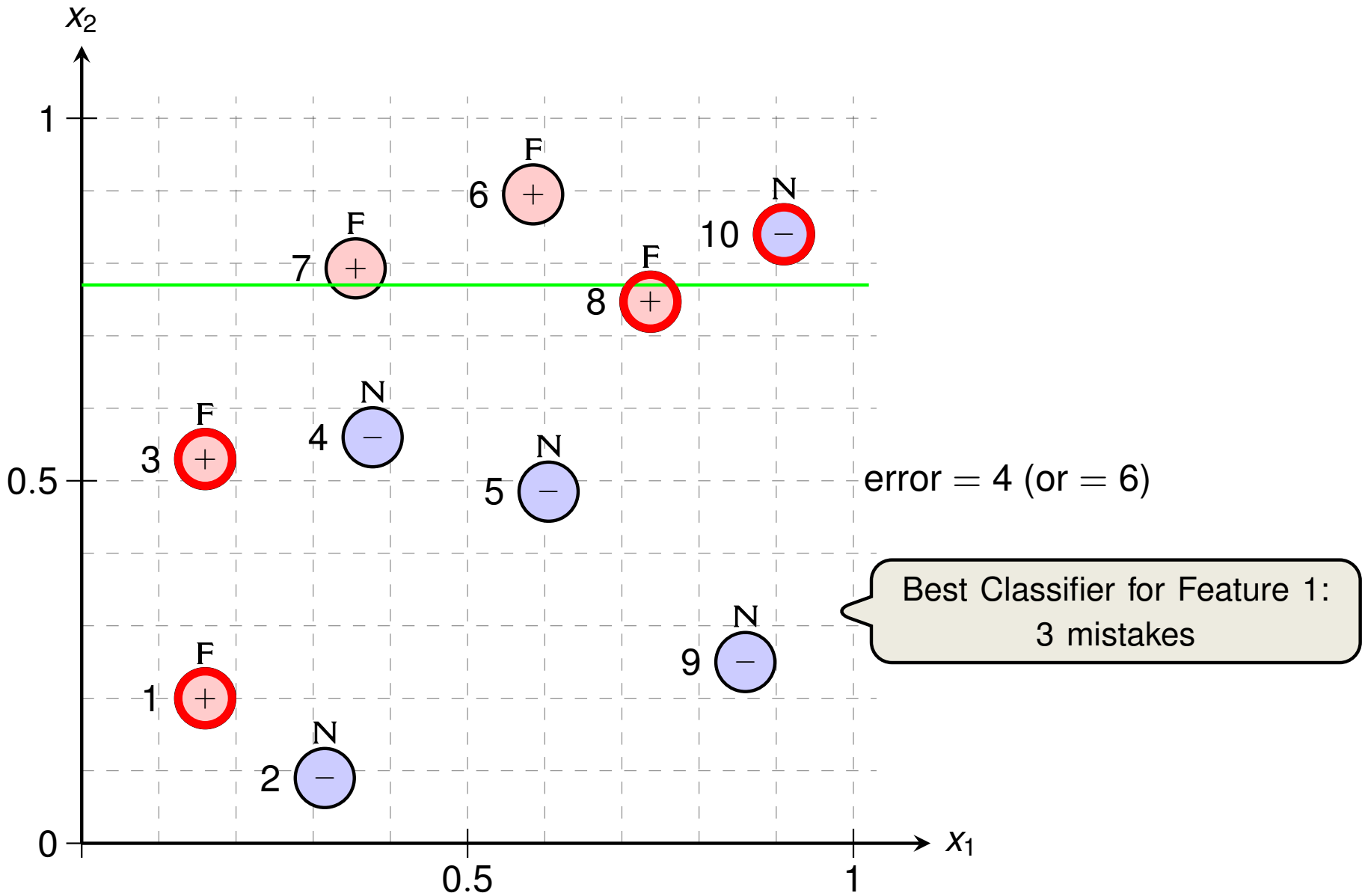
Schematic Example (3/3)



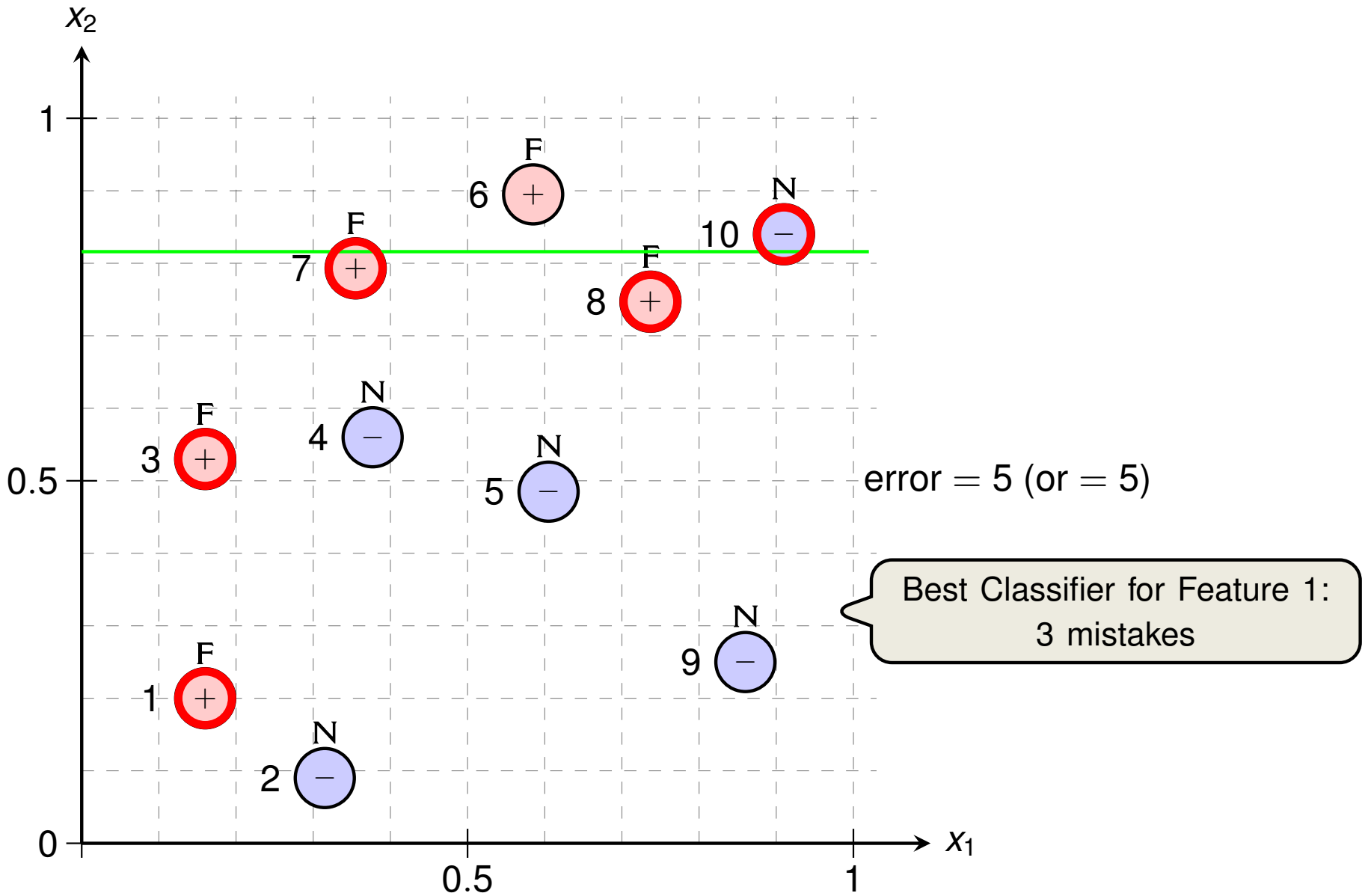
Schematic Example (3/3)



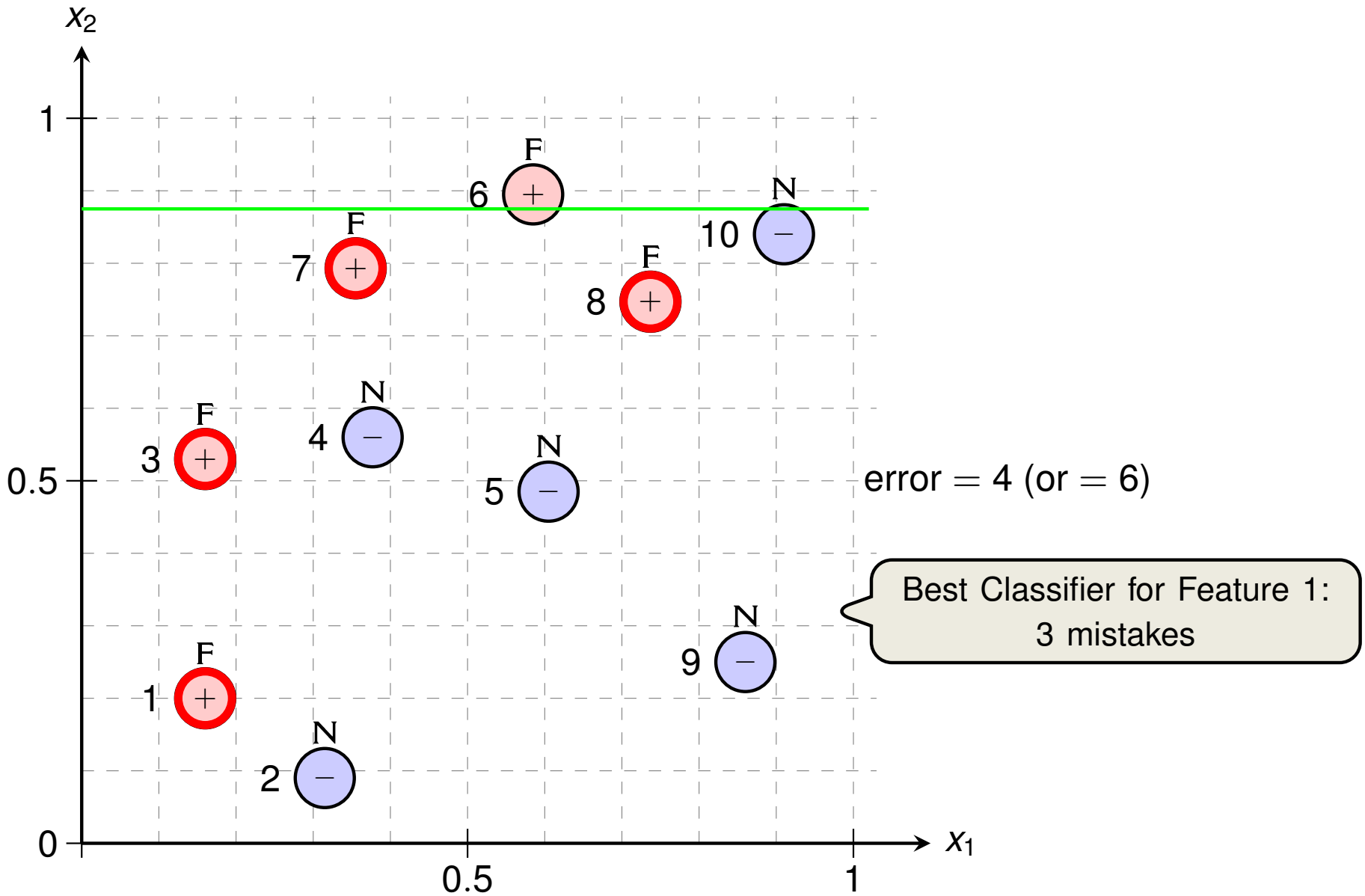
Schematic Example (3/3)



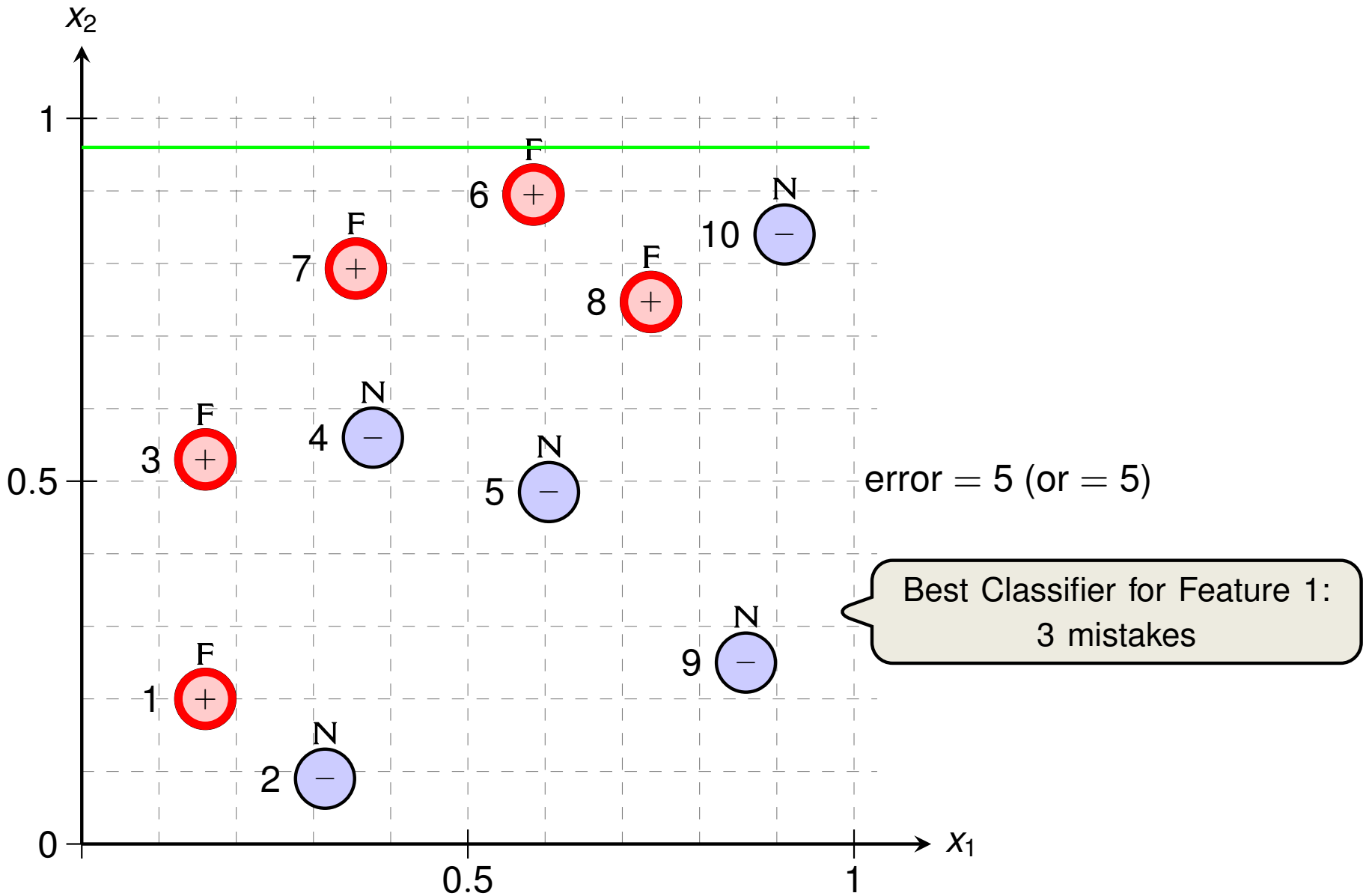
Schematic Example (3/3)



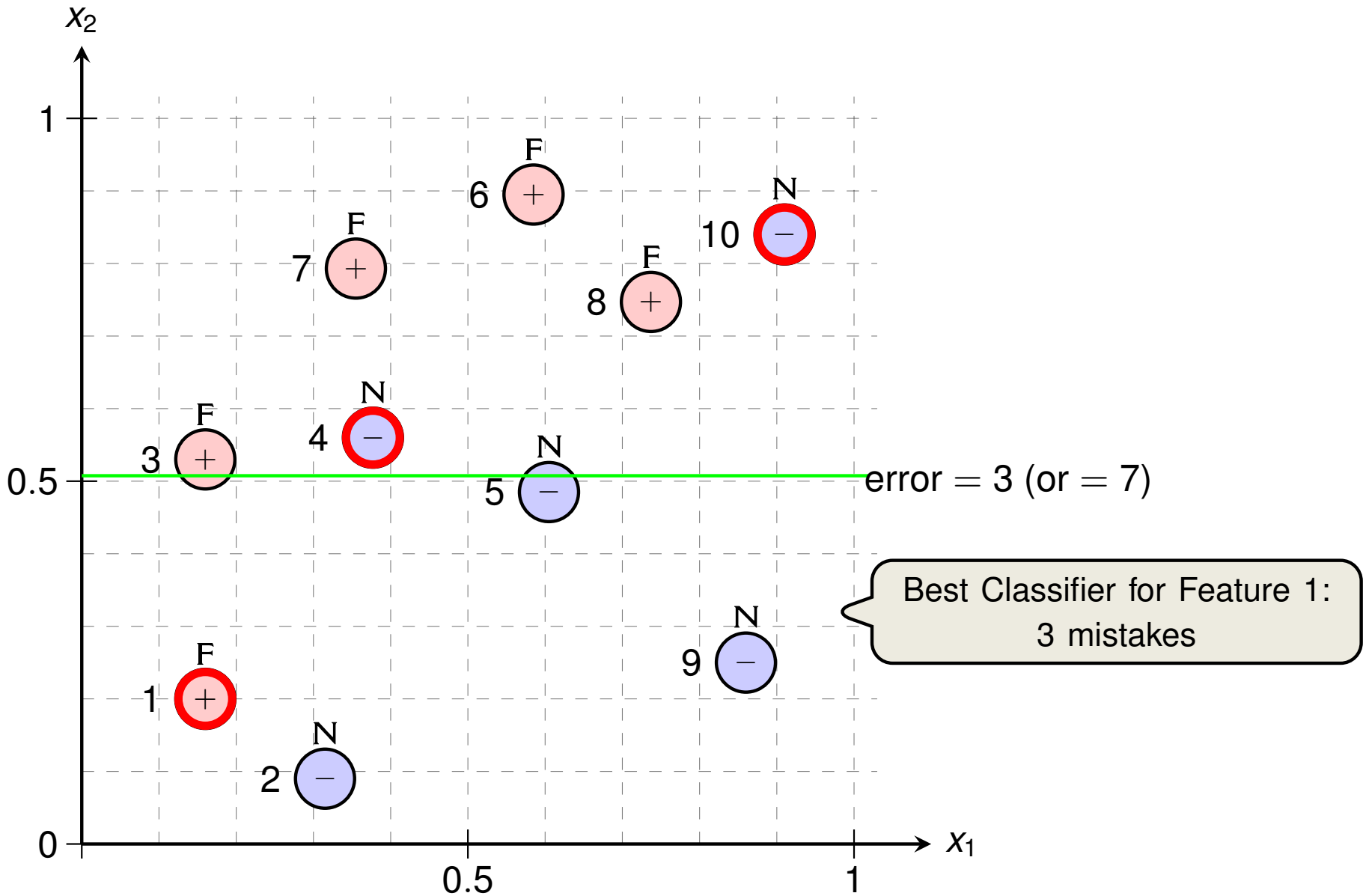
Schematic Example (3/3)



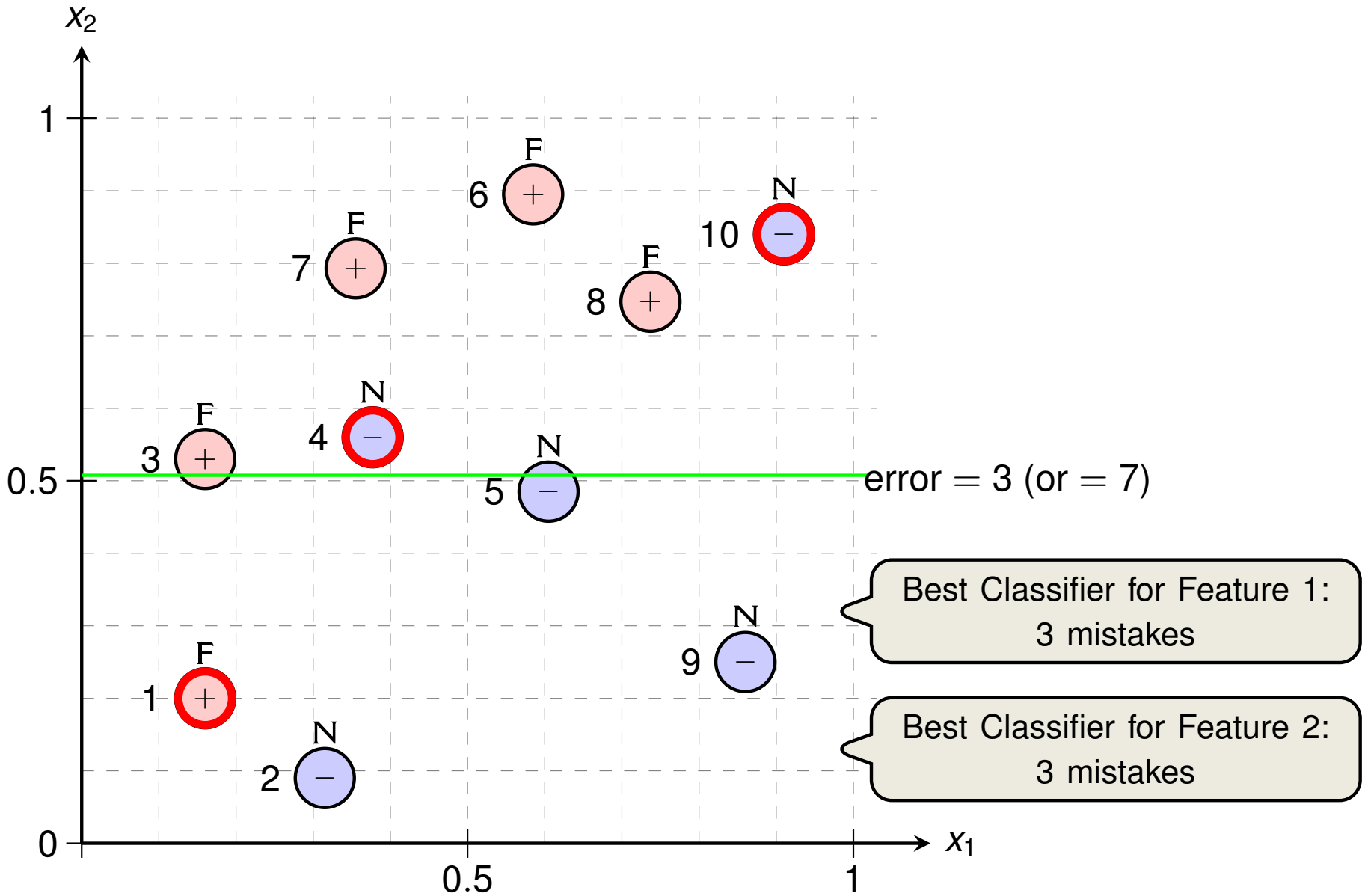
Schematic Example (3/3)



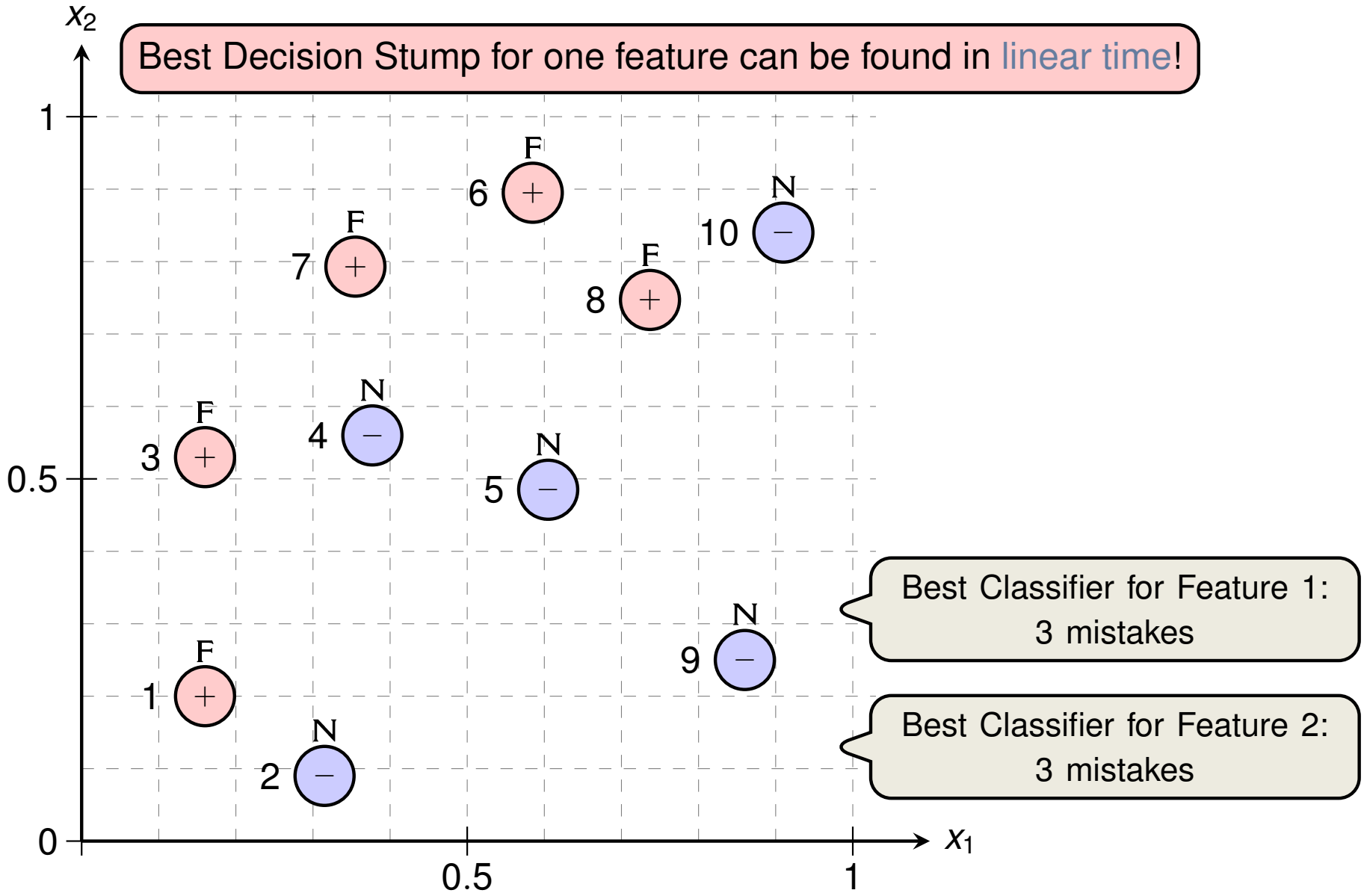
Schematic Example (3/3)



Schematic Example (3/3)



Schematic Example (3/3)



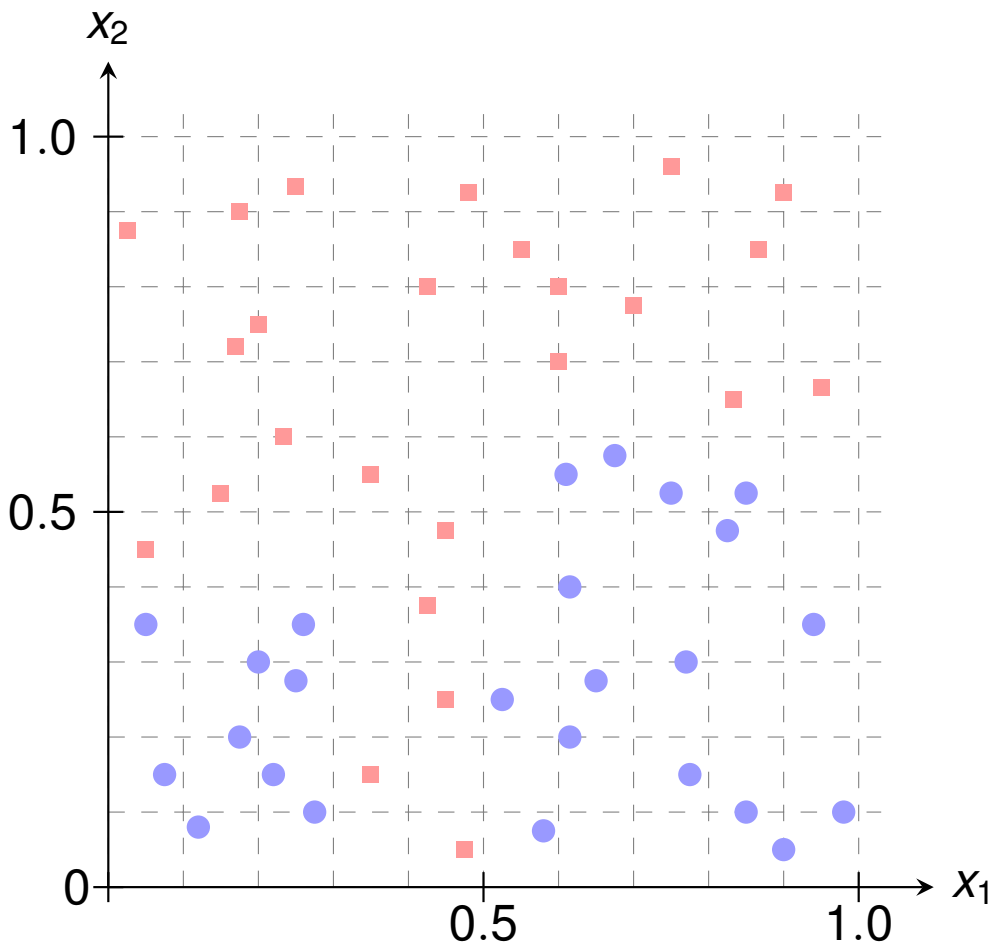
Outline

Illustrative Example

- Suppose we are given 50 classified data points (25 positive, 25 negative)

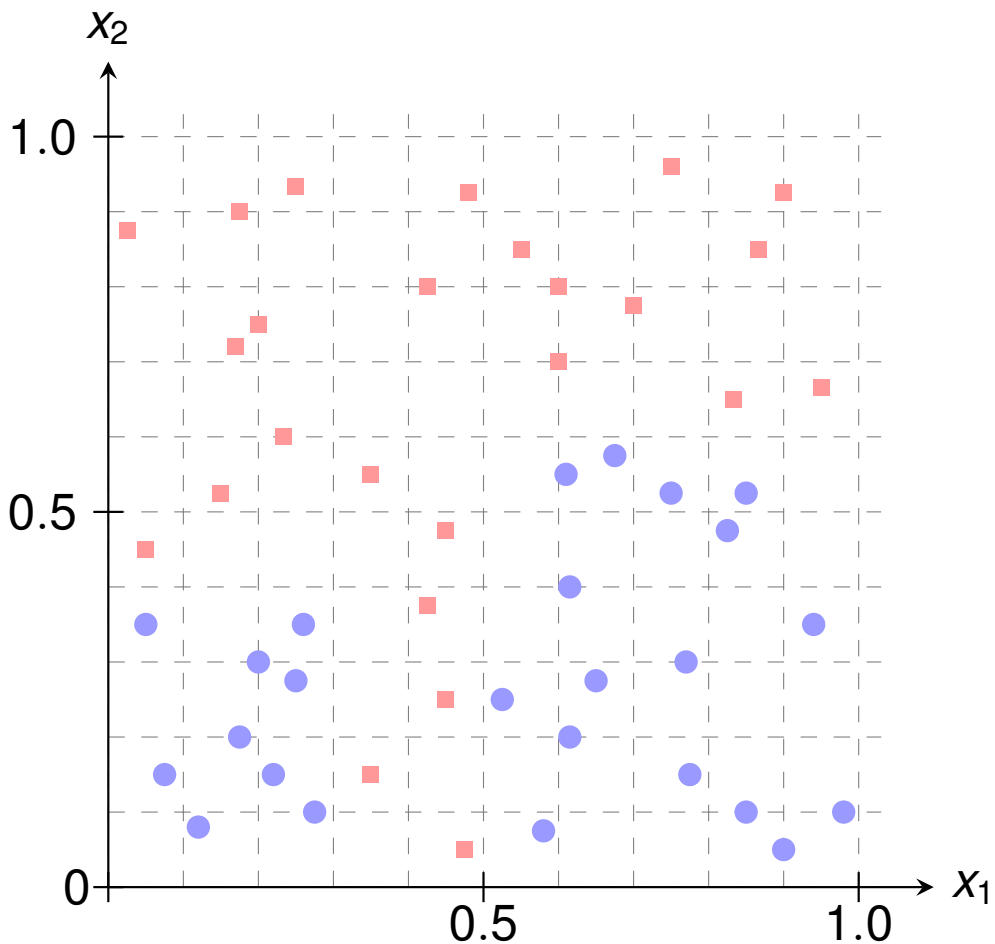
Illustrative Example

- Suppose we are given 50 classified data points (25 positive, 25 negative)



Illustrative Example

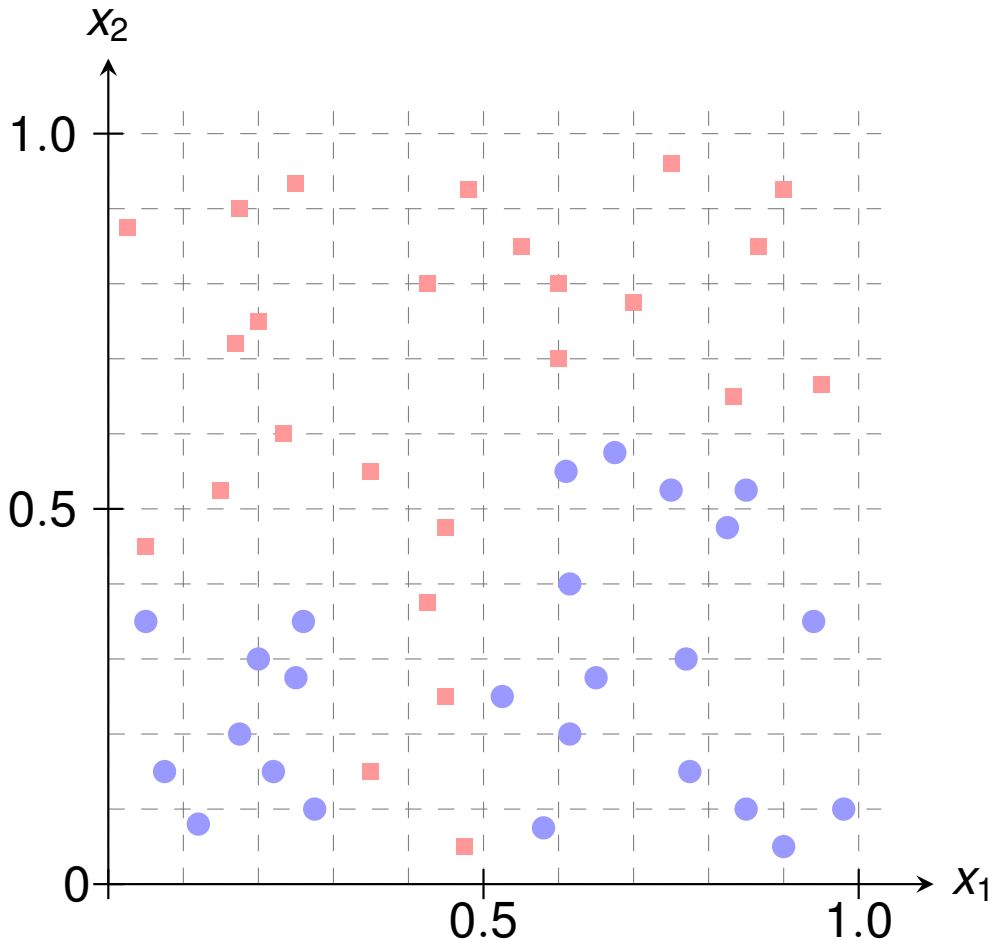
- Suppose we are given 50 classified data points (25 positive, 25 negative)
- Goal: Find a “simple” decision rule that classifies all points correctly



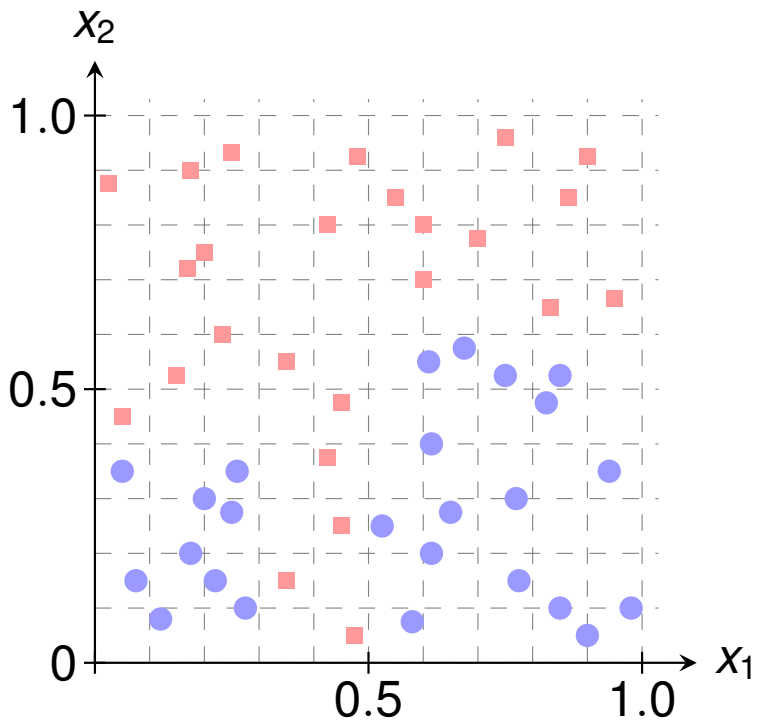
Illustrative Example

- Suppose we are given 50 classified data points (25 positive, 25 negative)
- Goal: Find a “simple” decision rule that classifies all points correctly

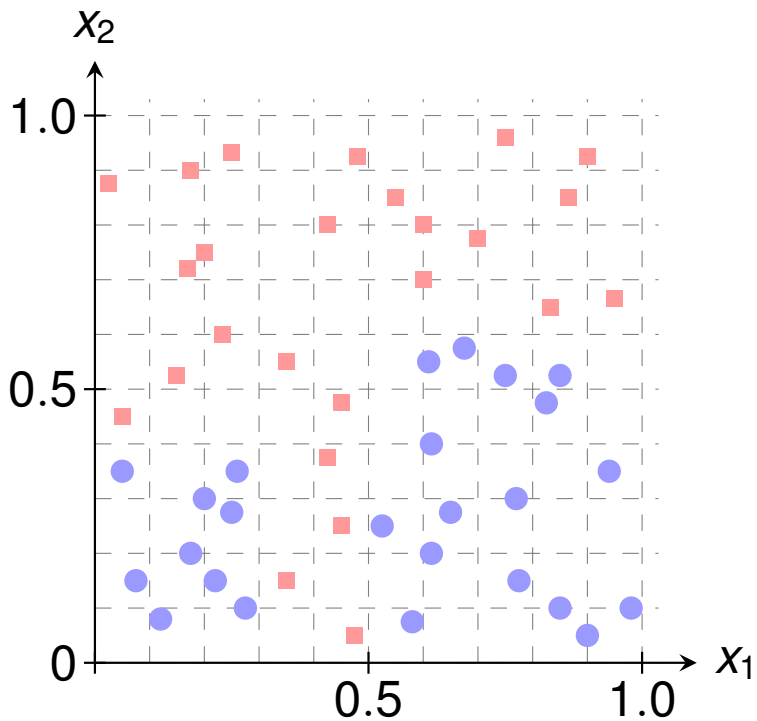
Challenge: A straight decision boundary (decision stump) does not work



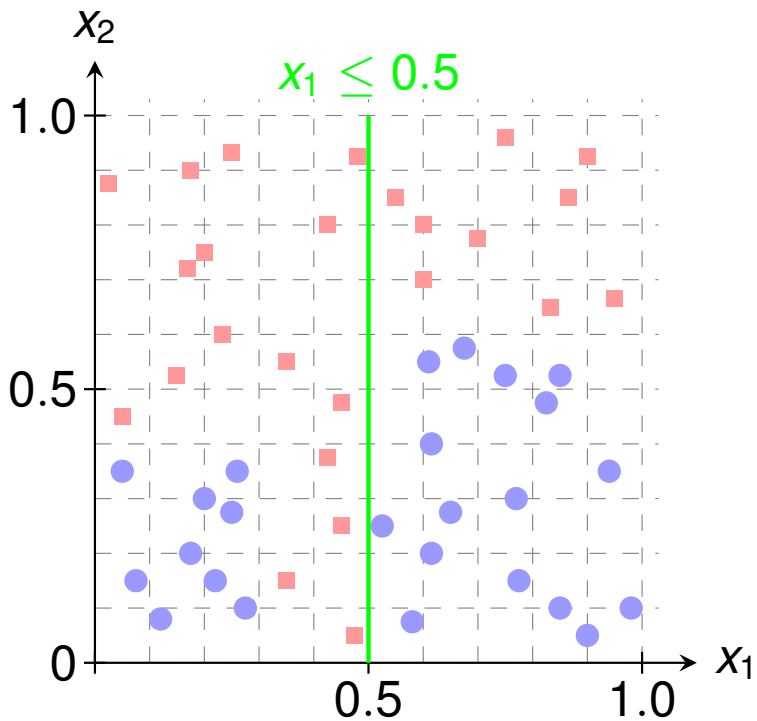
Intro: An Example of building a Decision Tree



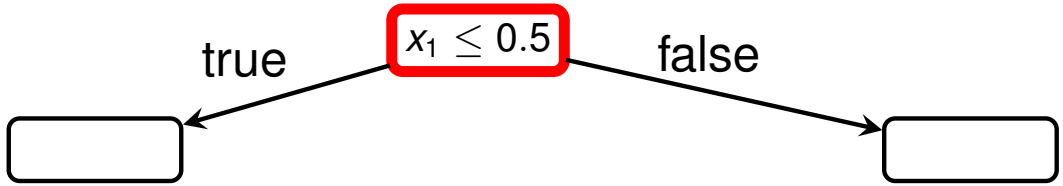
Intro: An Example of building a Decision Tree

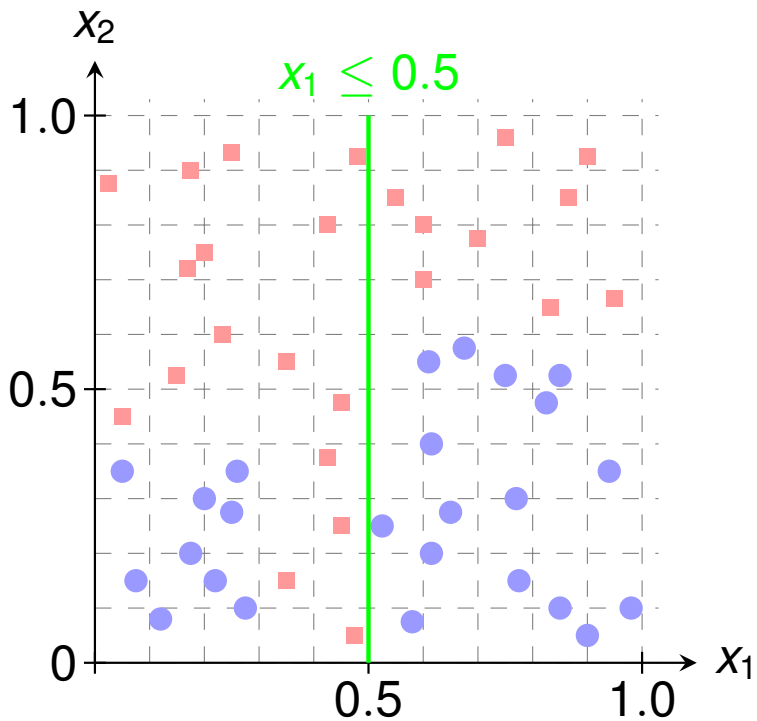


Intro: An Example of building a Decision Tree

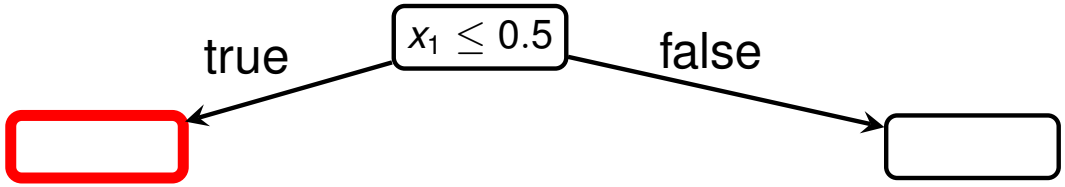


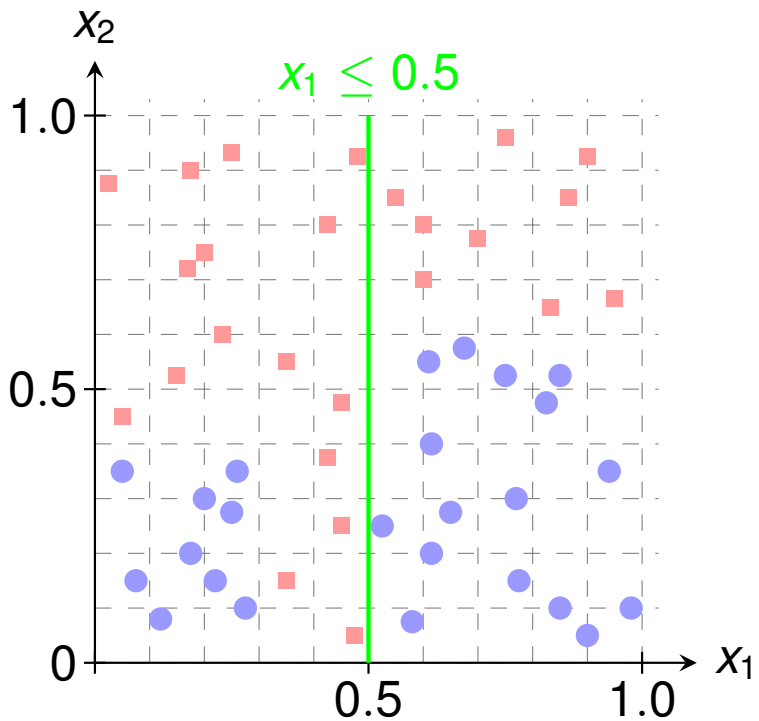
Intro: An Example of building a Decision Tree



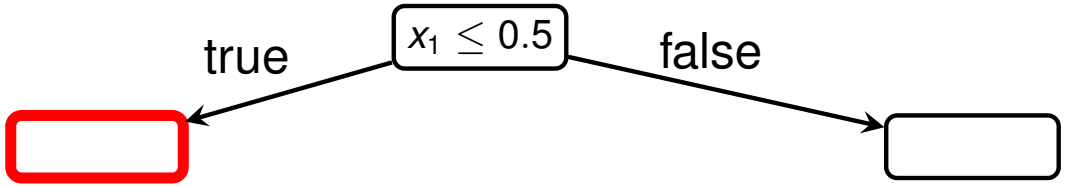


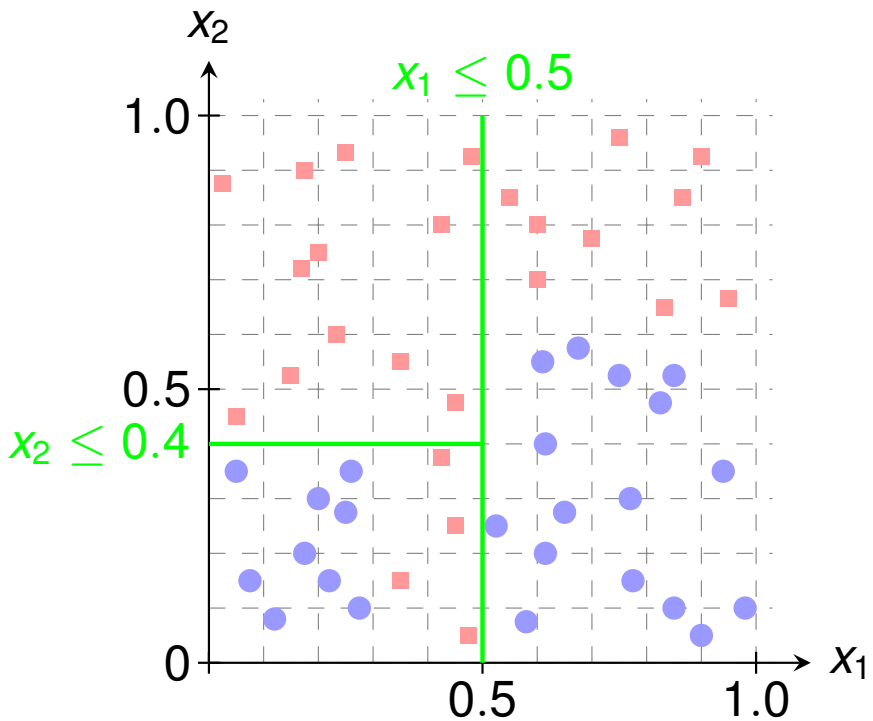
Intro: An Example of building a Decision Tree



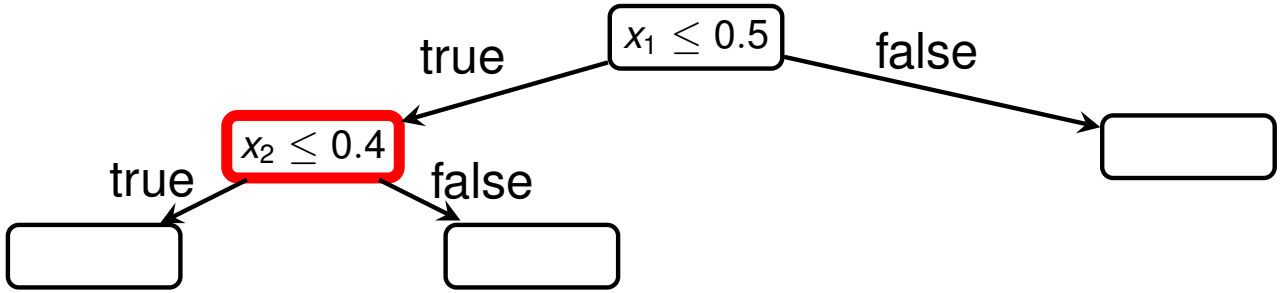


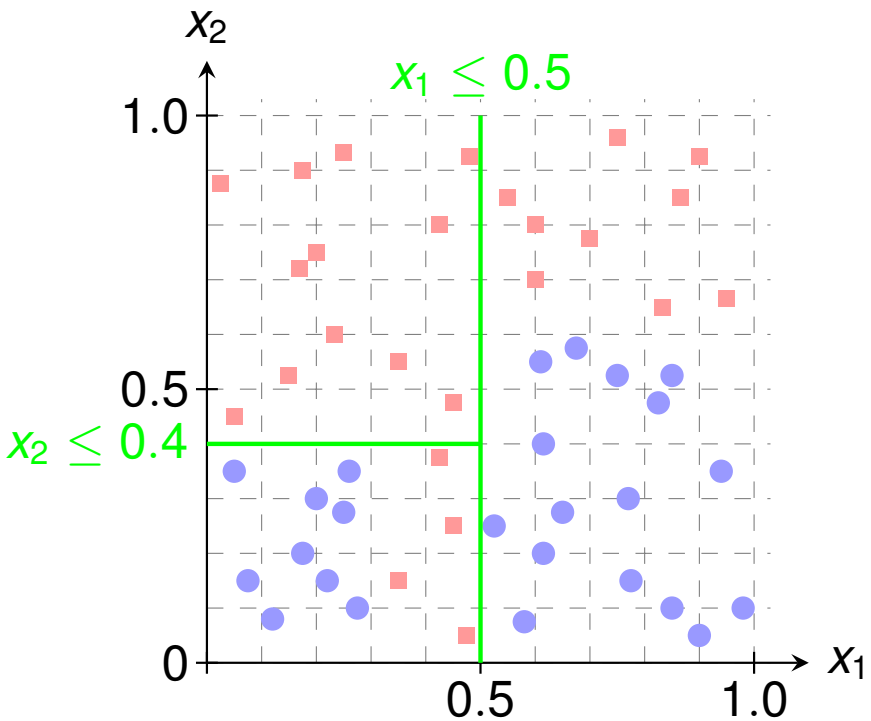
Intro: An Example of building a Decision Tree



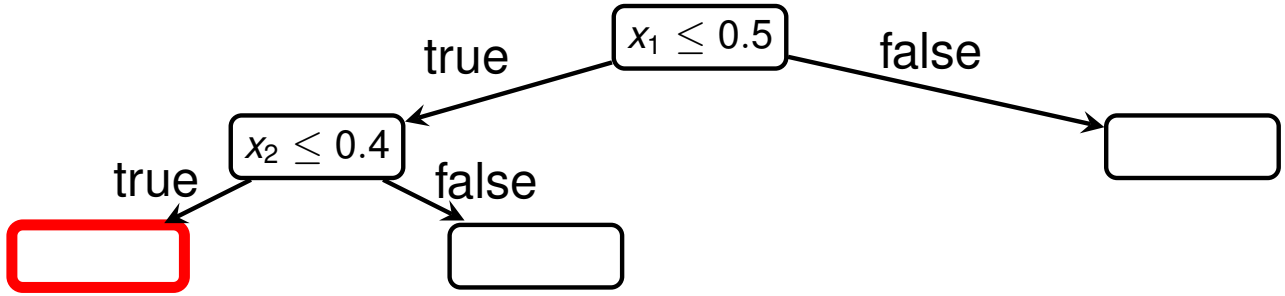


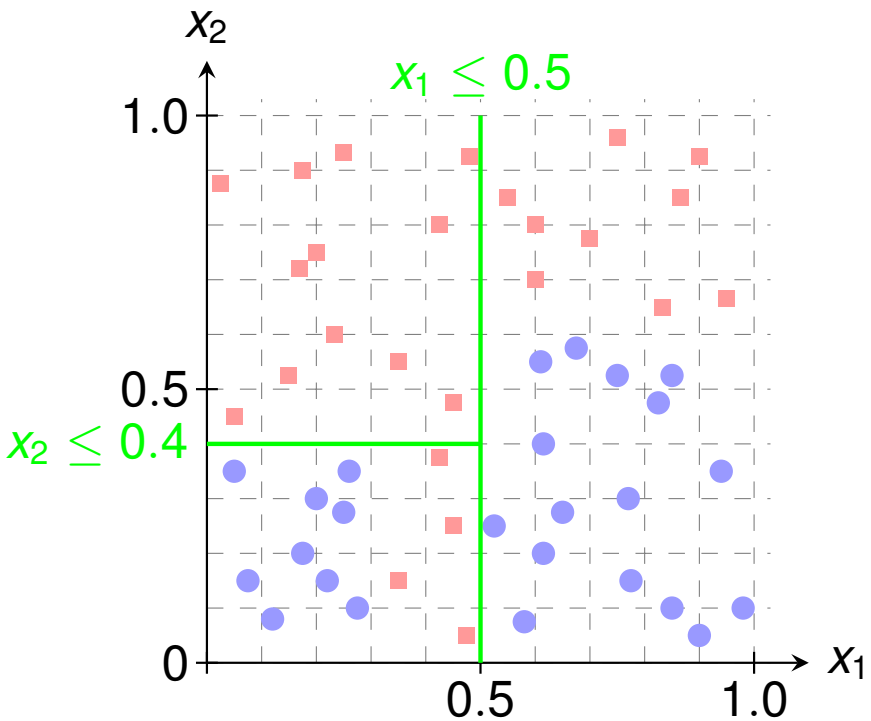
Intro: An Example of building a Decision Tree



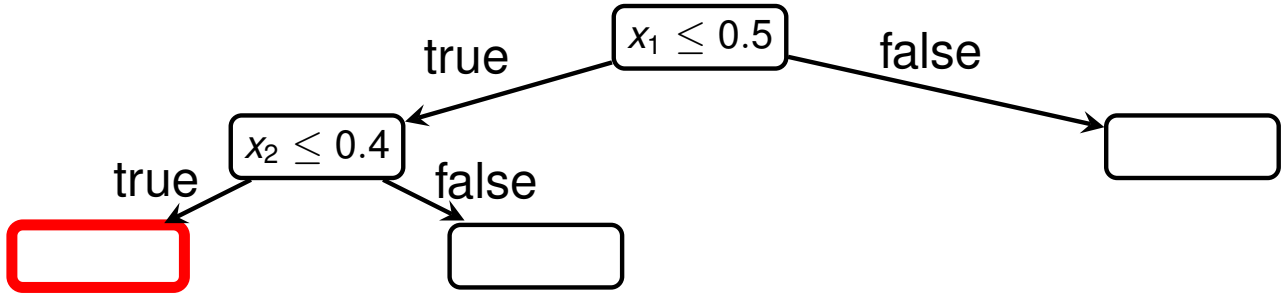


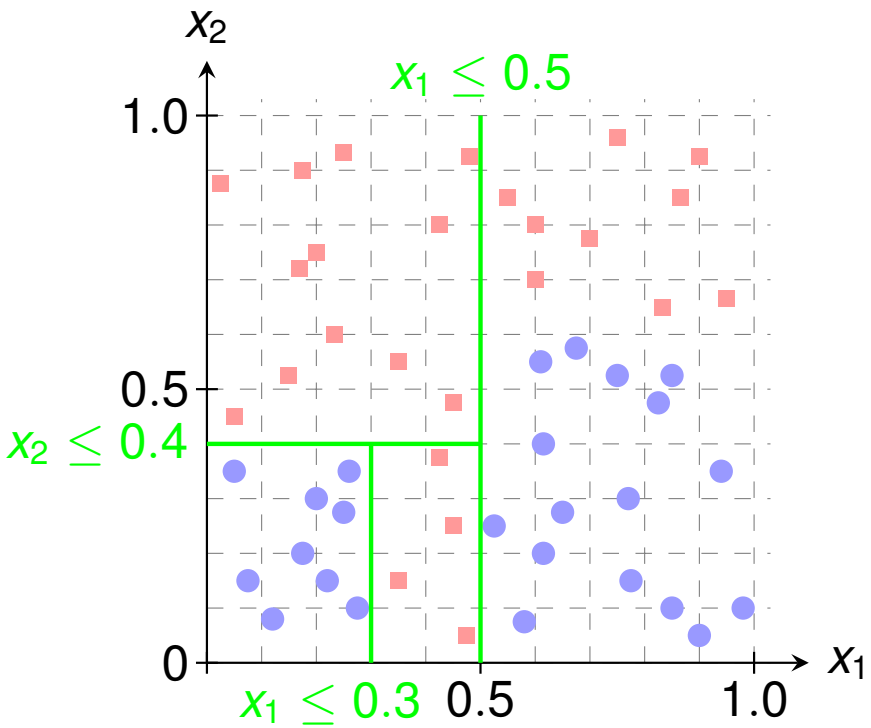
Intro: An Example of building a Decision Tree



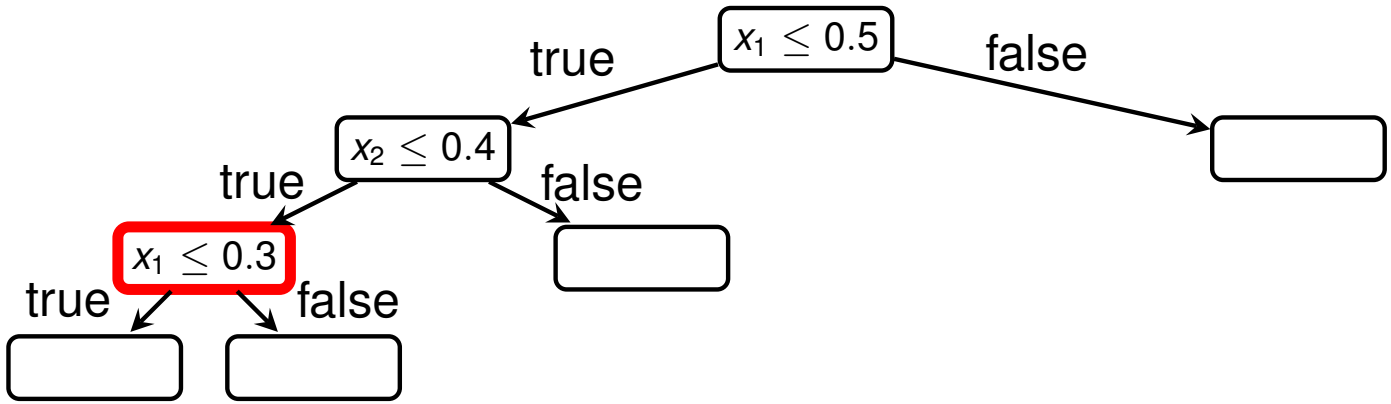


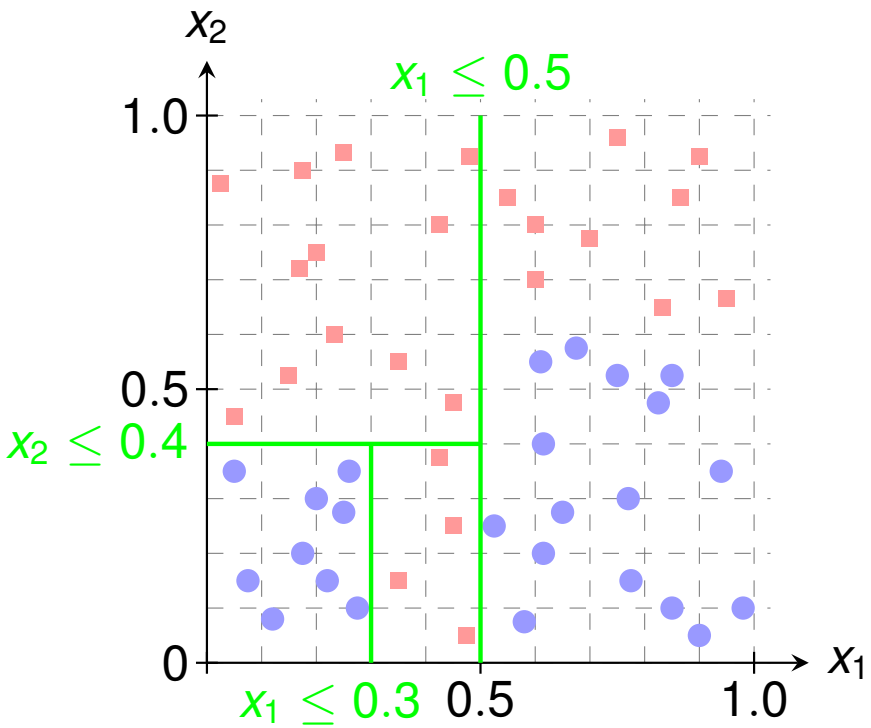
Intro: An Example of building a Decision Tree



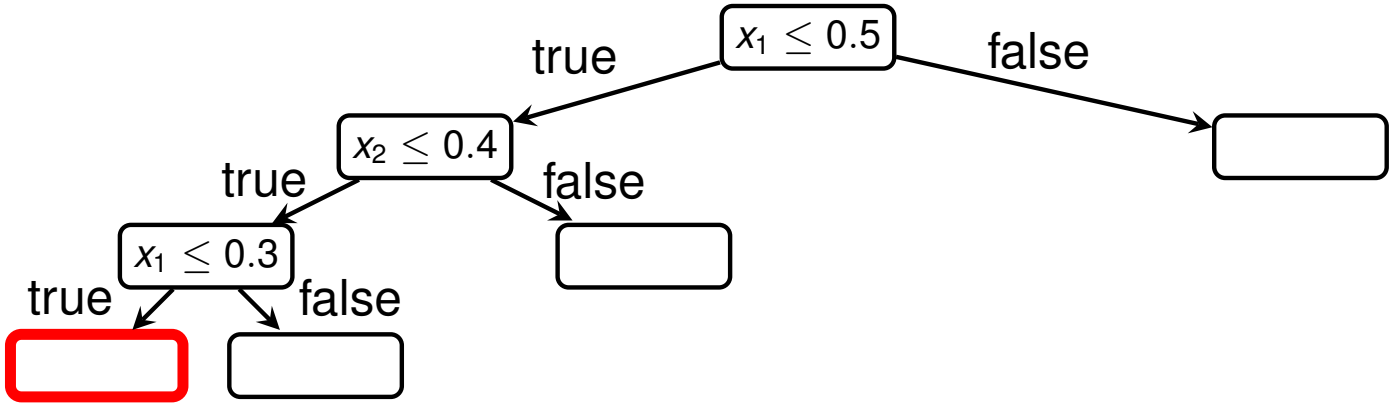


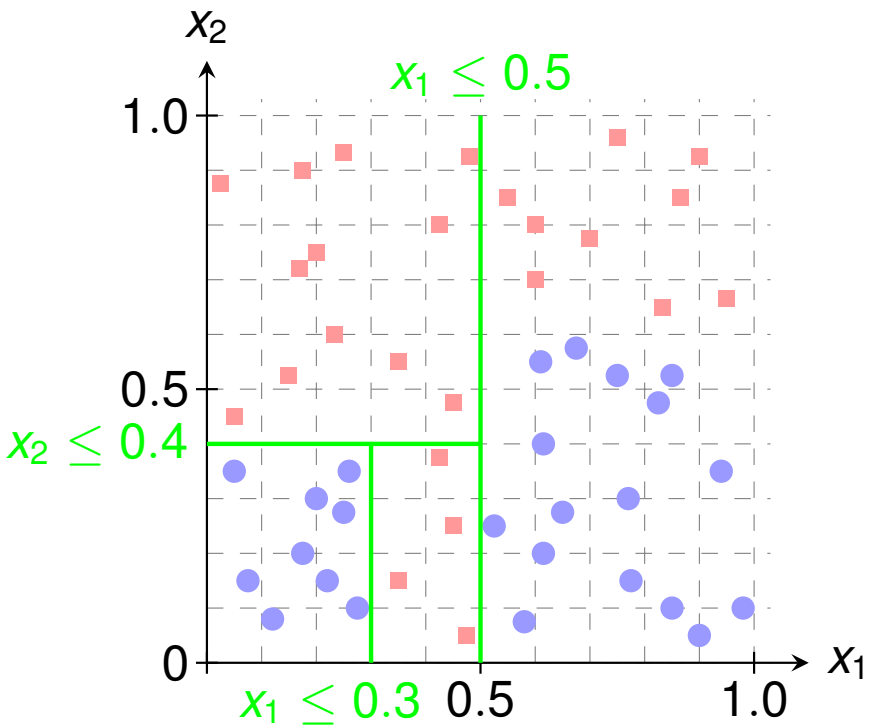
Intro: An Example of building a Decision Tree



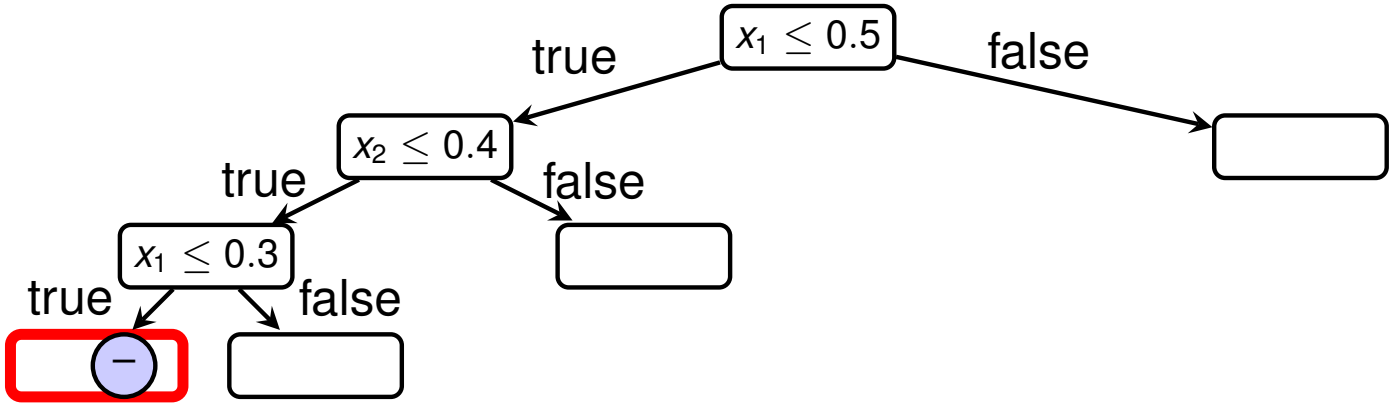


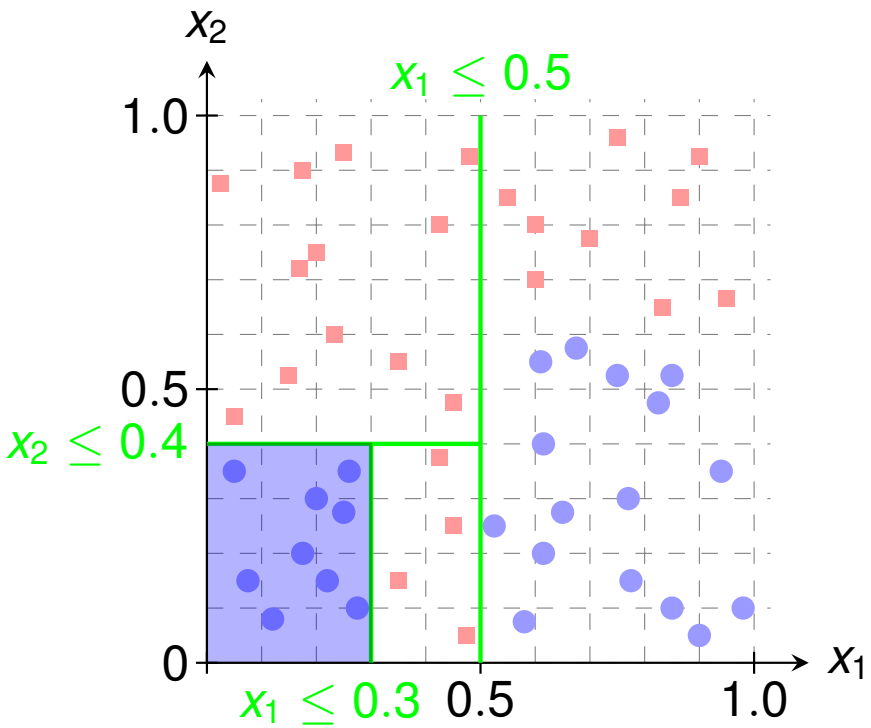
Intro: An Example of building a Decision Tree



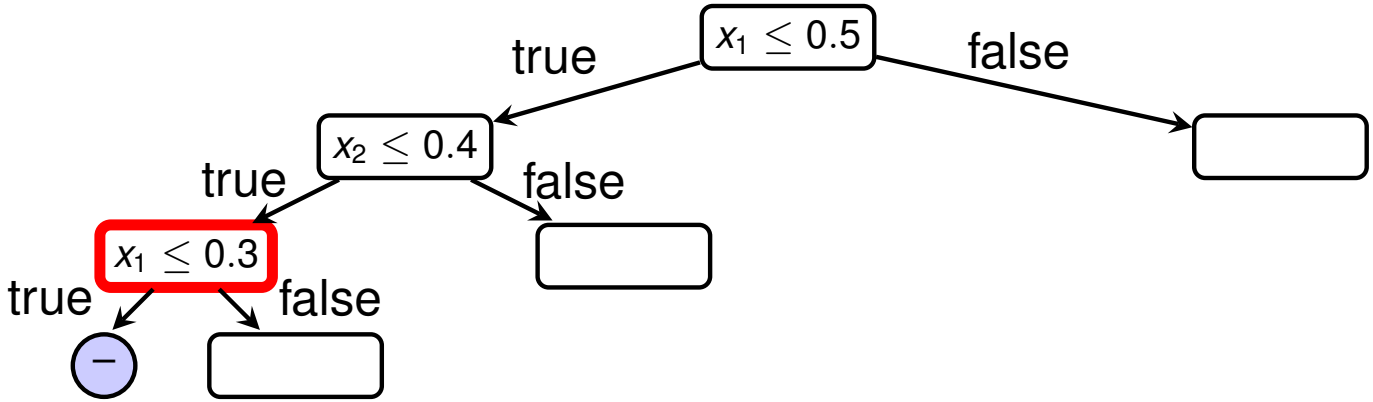


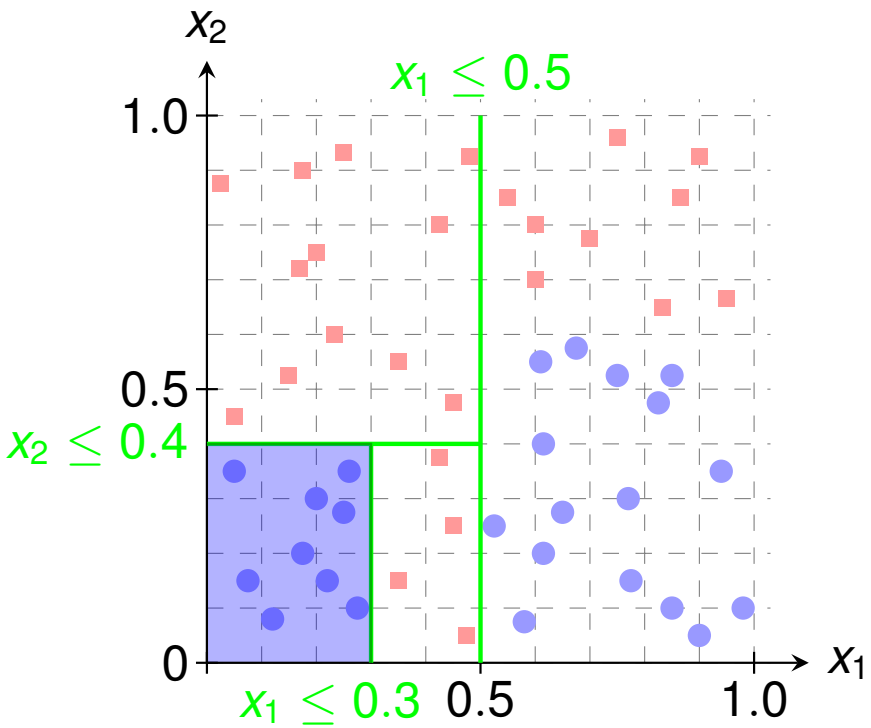
Intro: An Example of building a Decision Tree



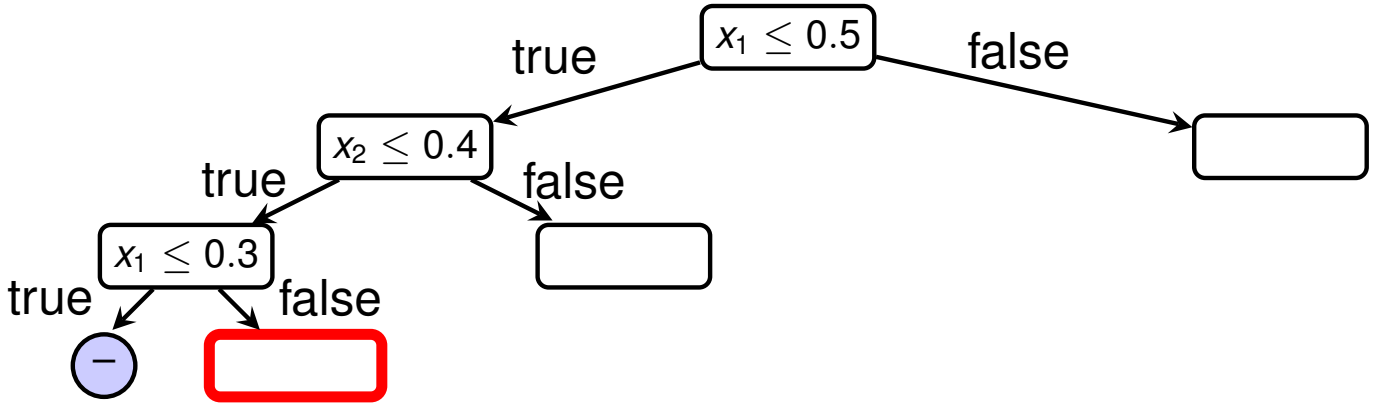


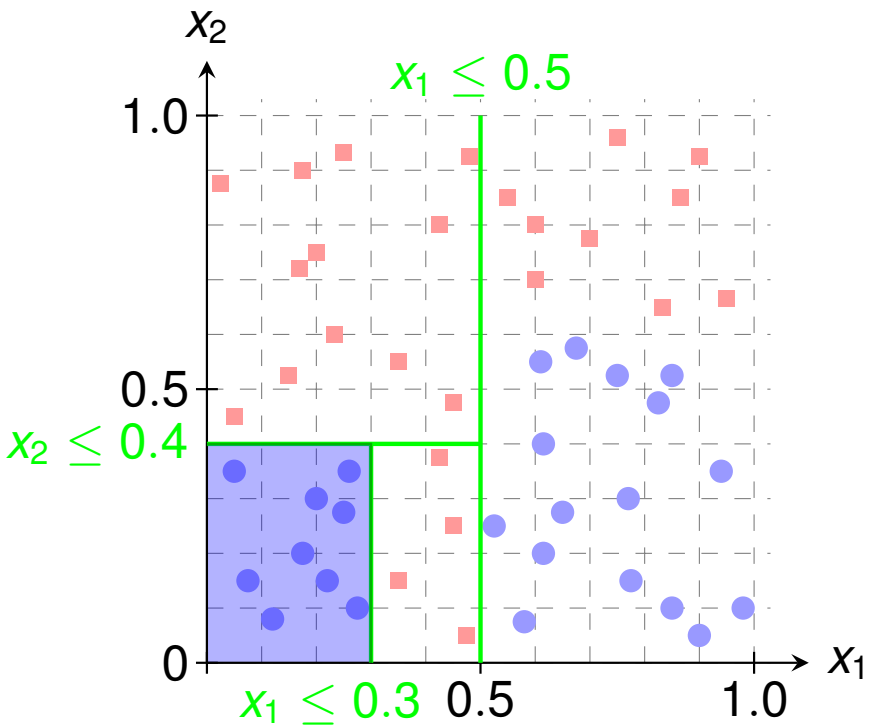
Intro: An Example of building a Decision Tree



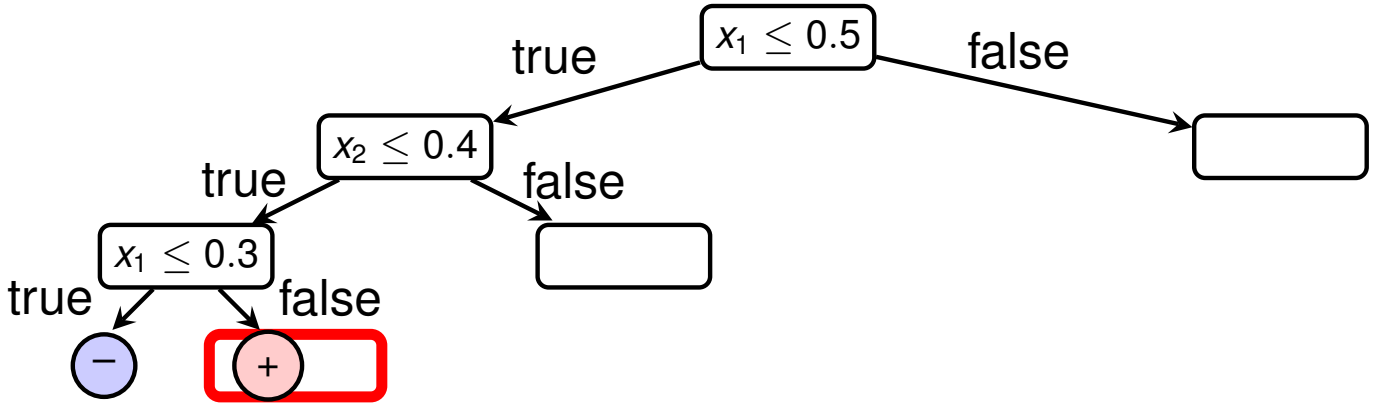


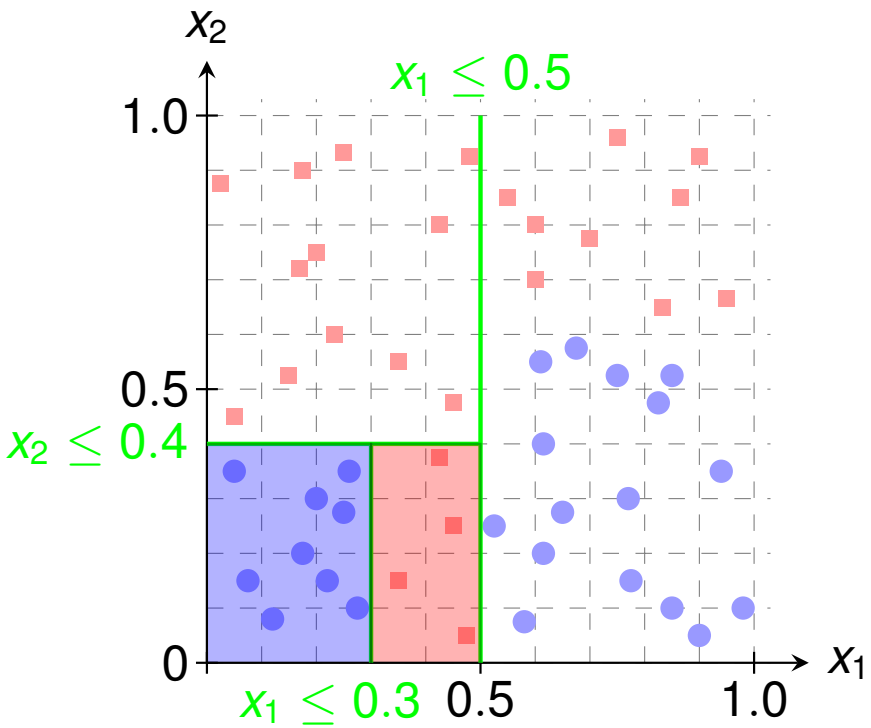
Intro: An Example of building a Decision Tree



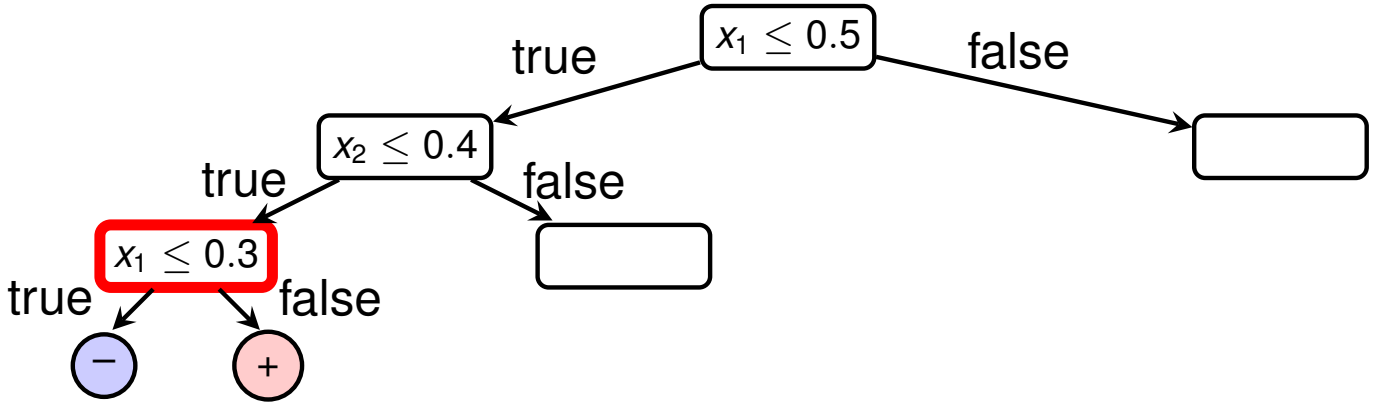


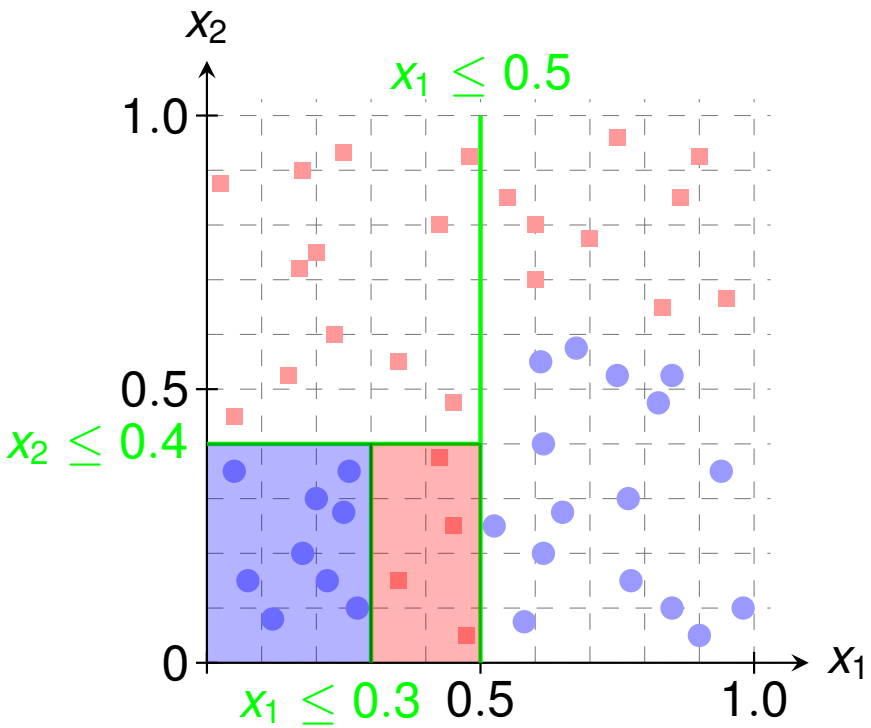
Intro: An Example of building a Decision Tree



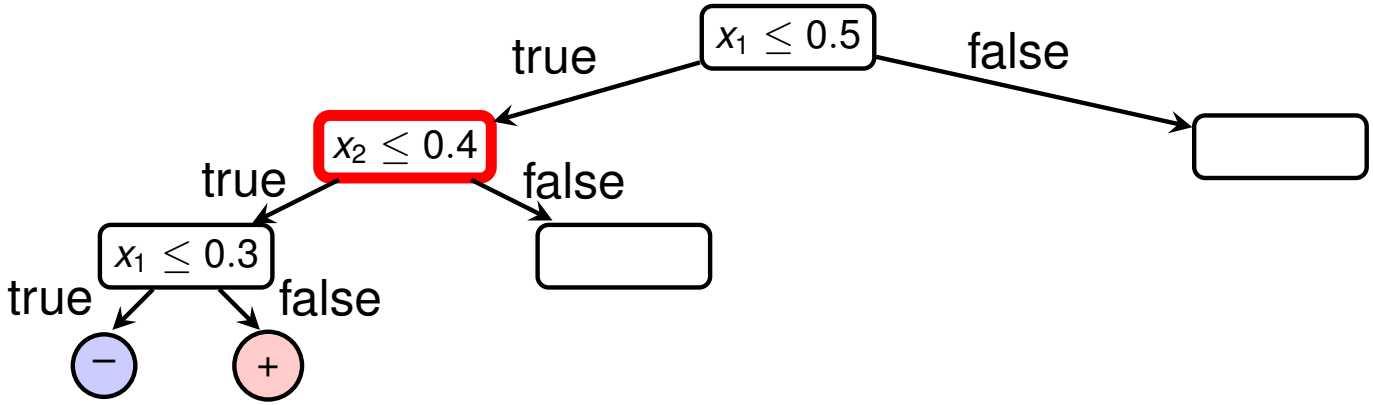


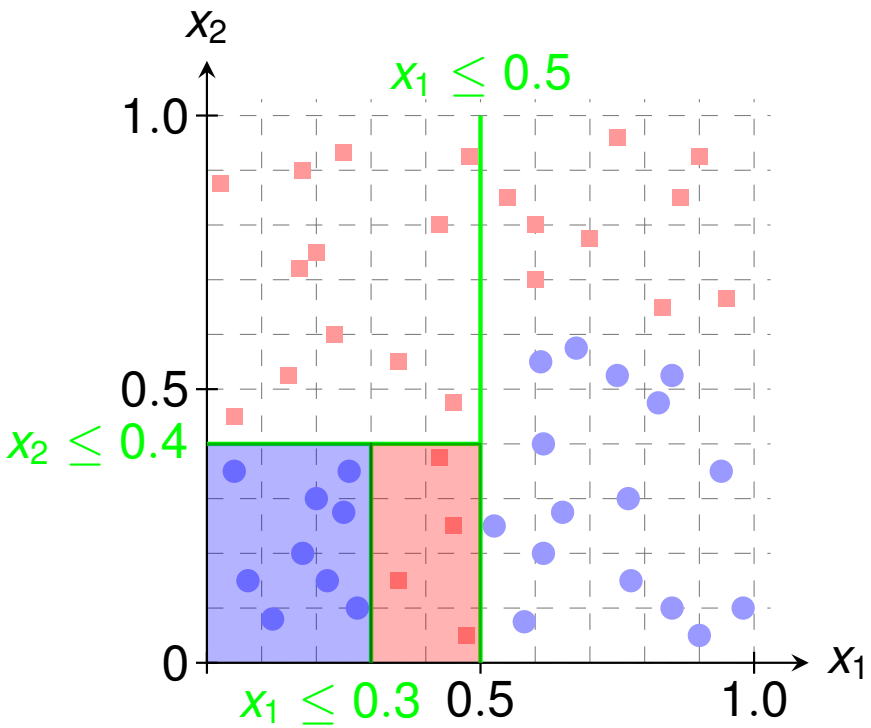
Intro: An Example of building a Decision Tree



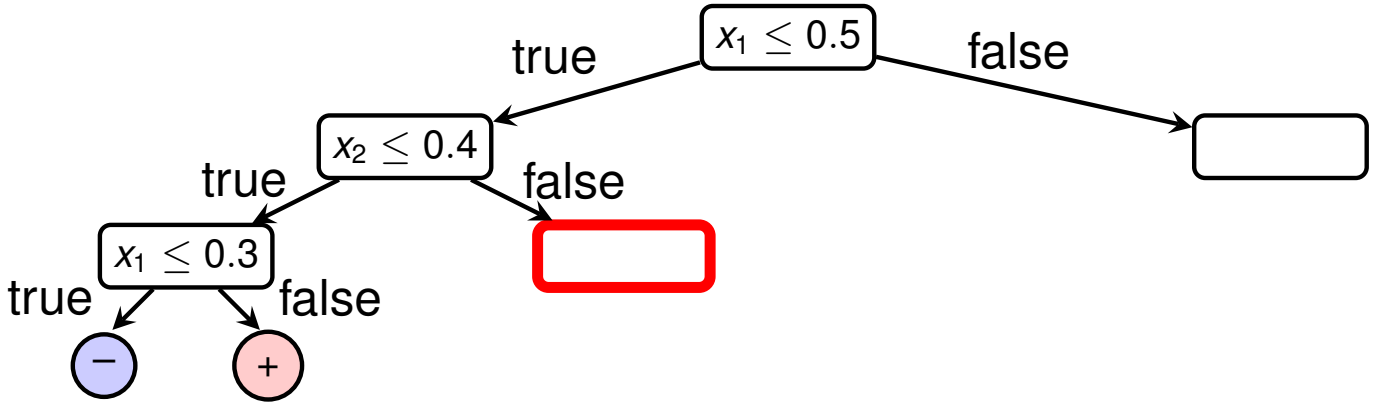


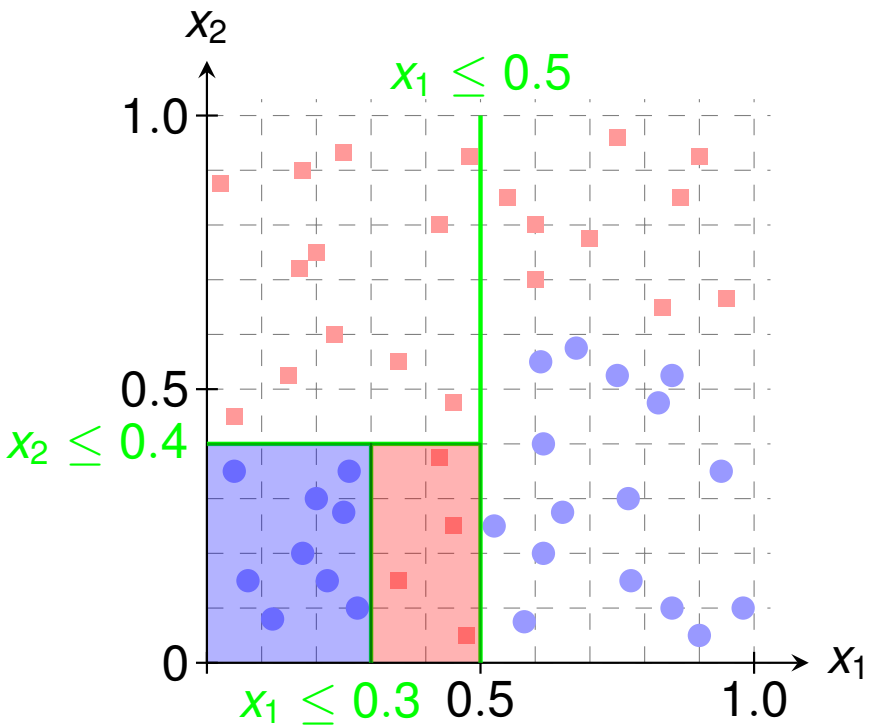
Intro: An Example of building a Decision Tree



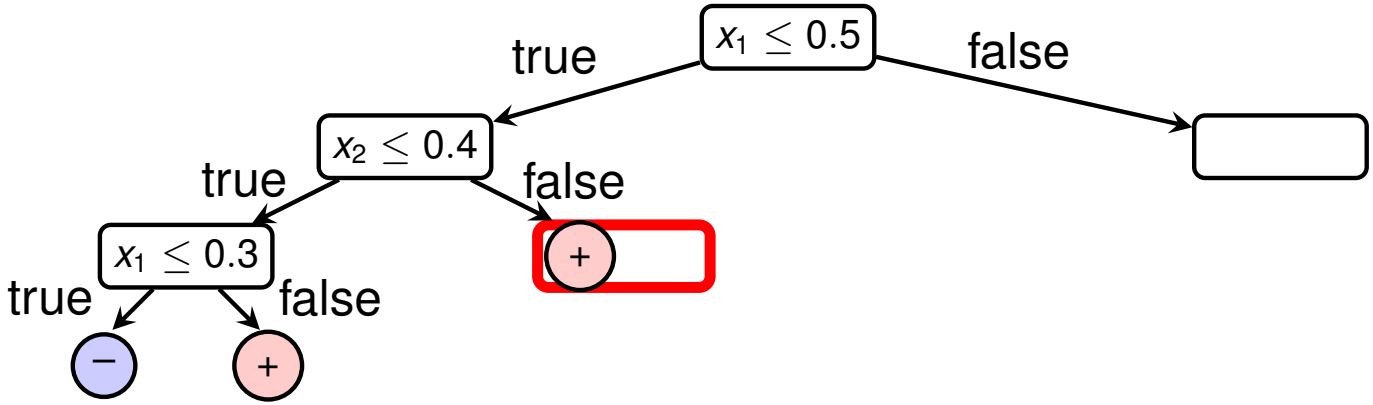


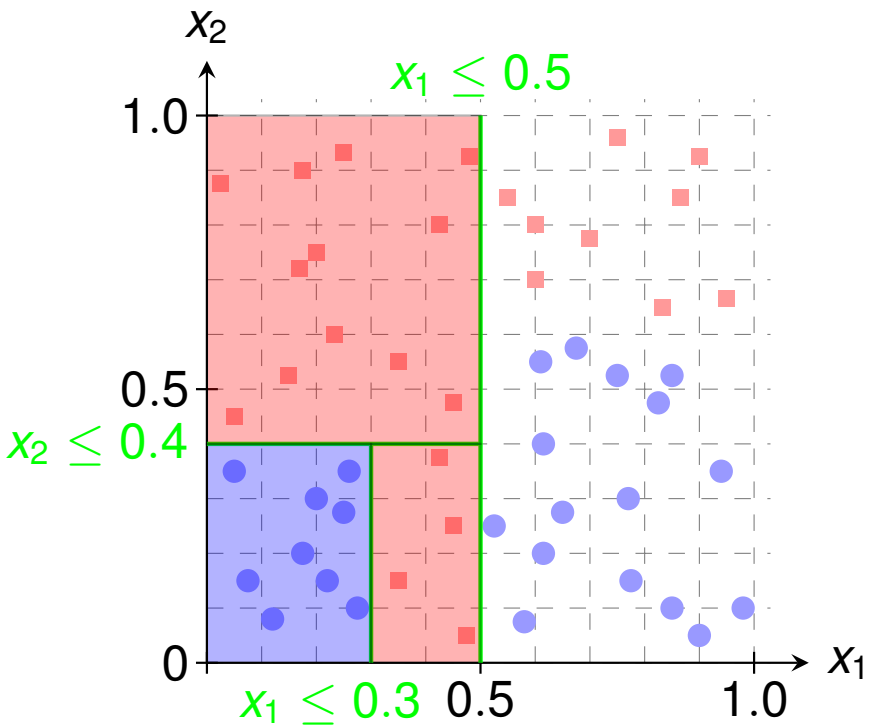
Intro: An Example of building a Decision Tree



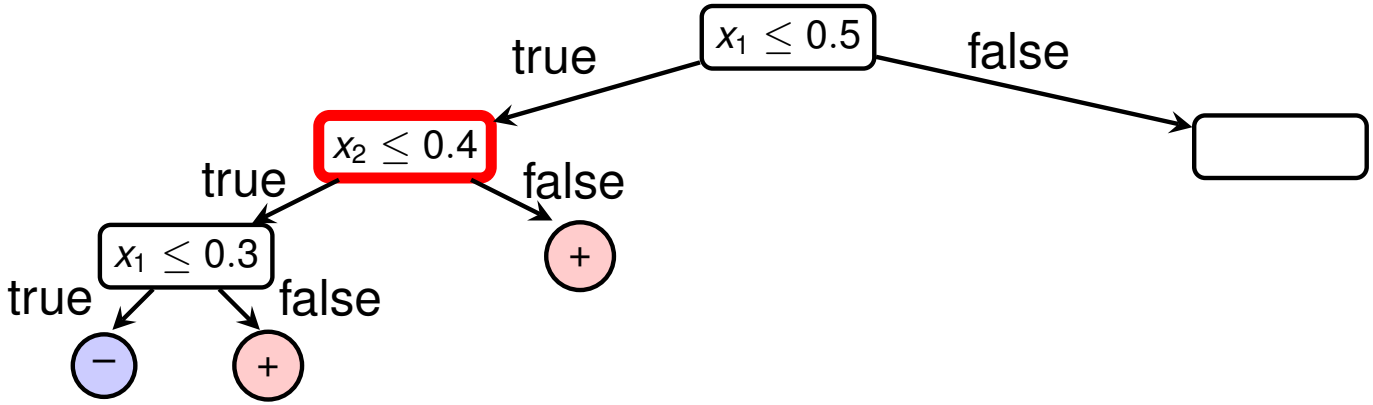


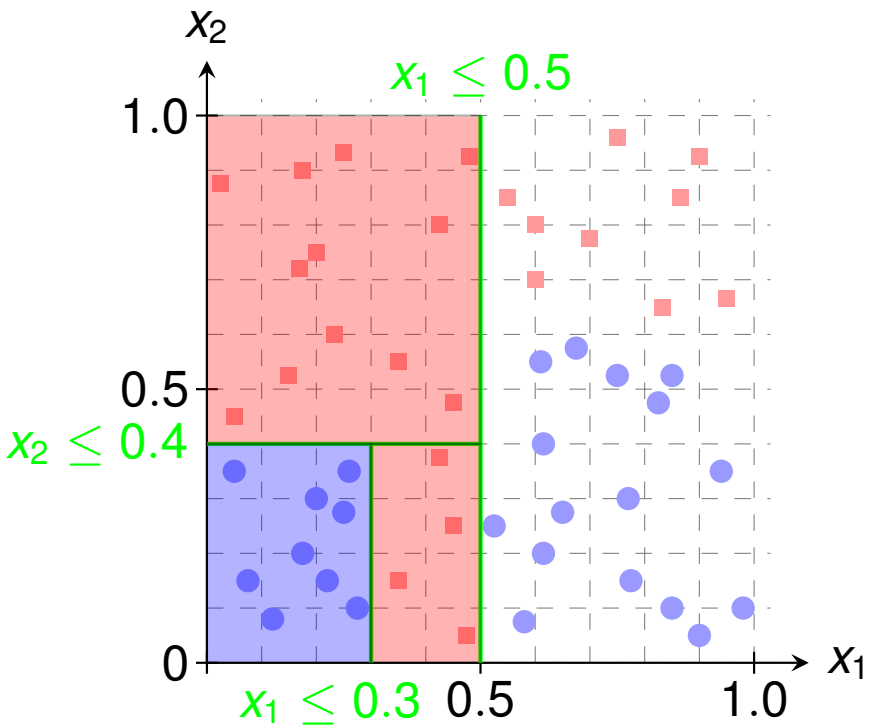
Intro: An Example of building a Decision Tree



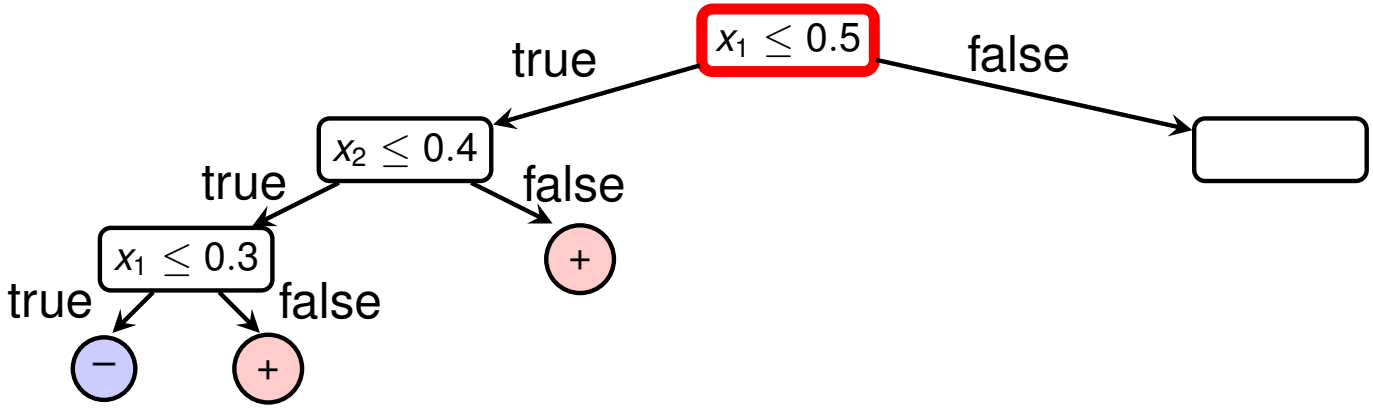


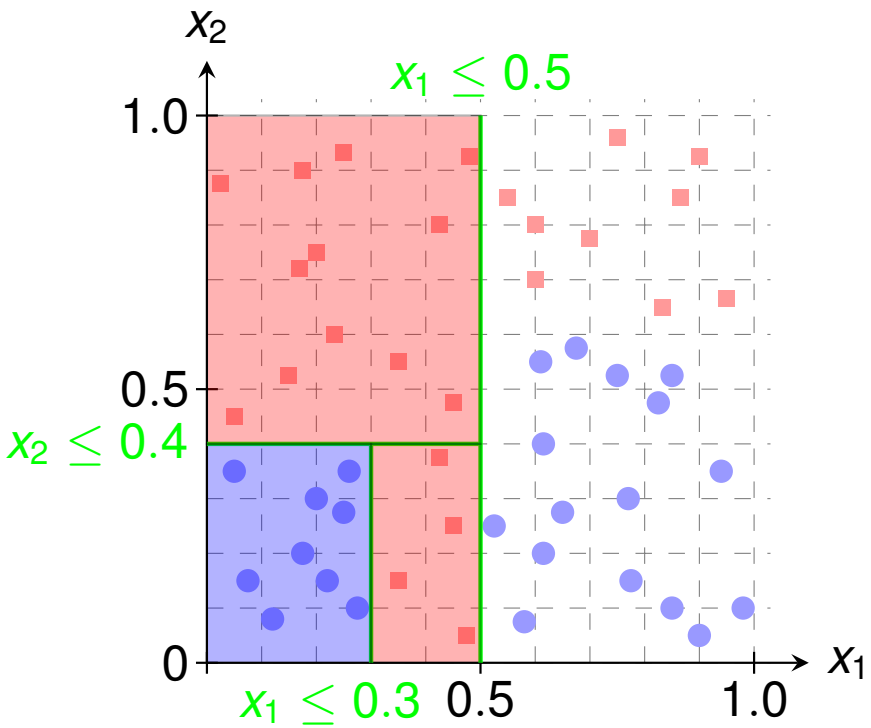
Intro: An Example of building a Decision Tree



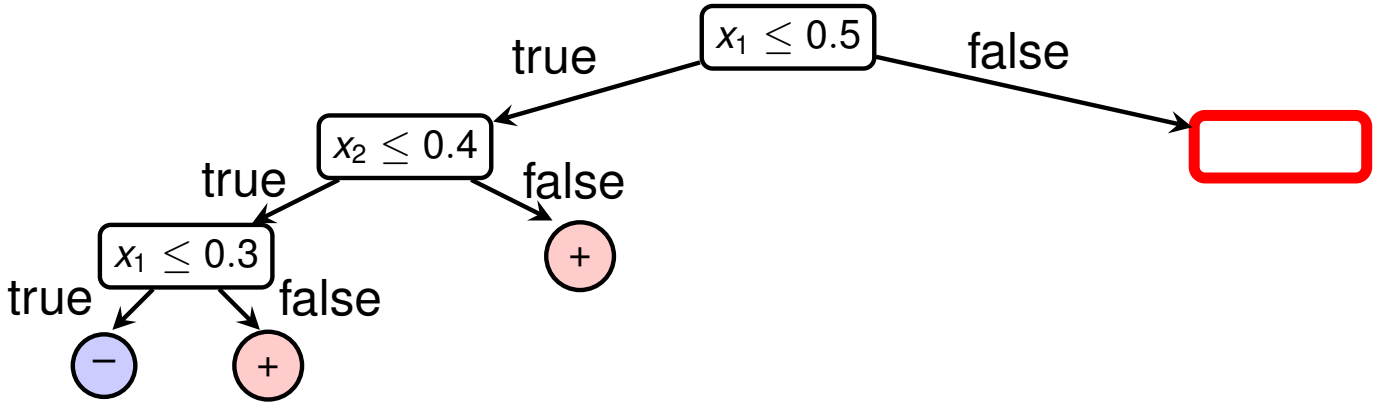


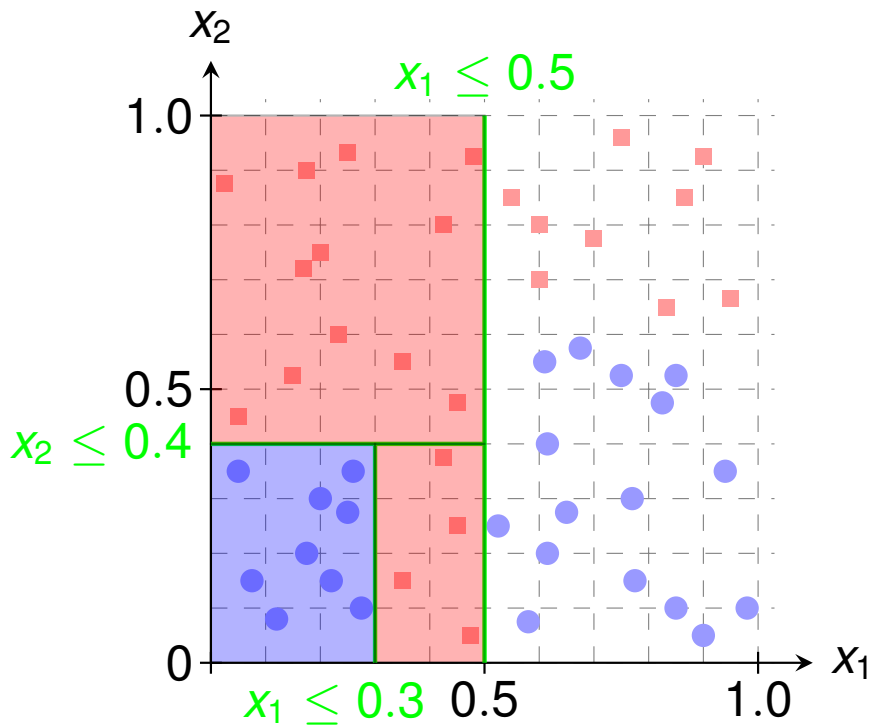
Intro: An Example of building a Decision Tree



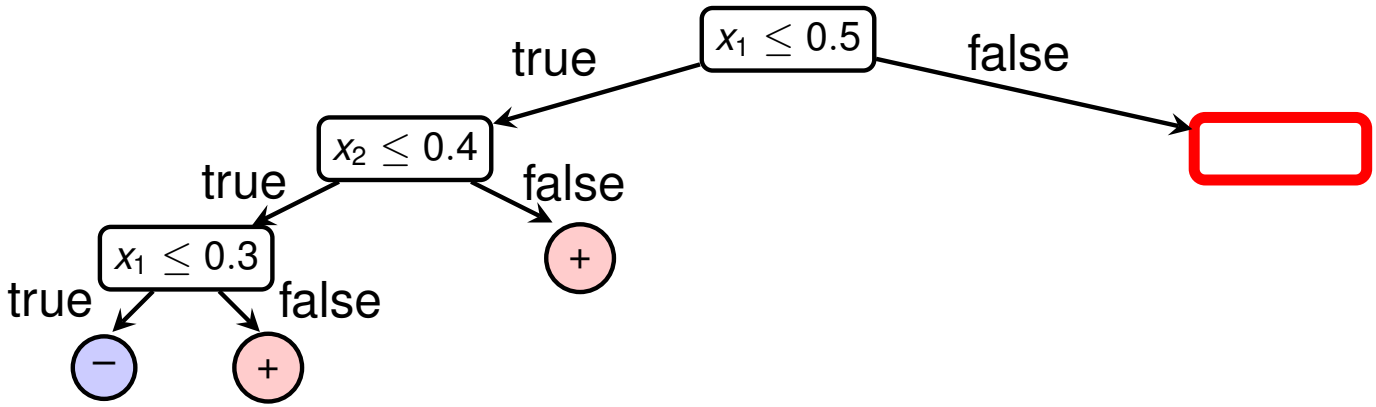


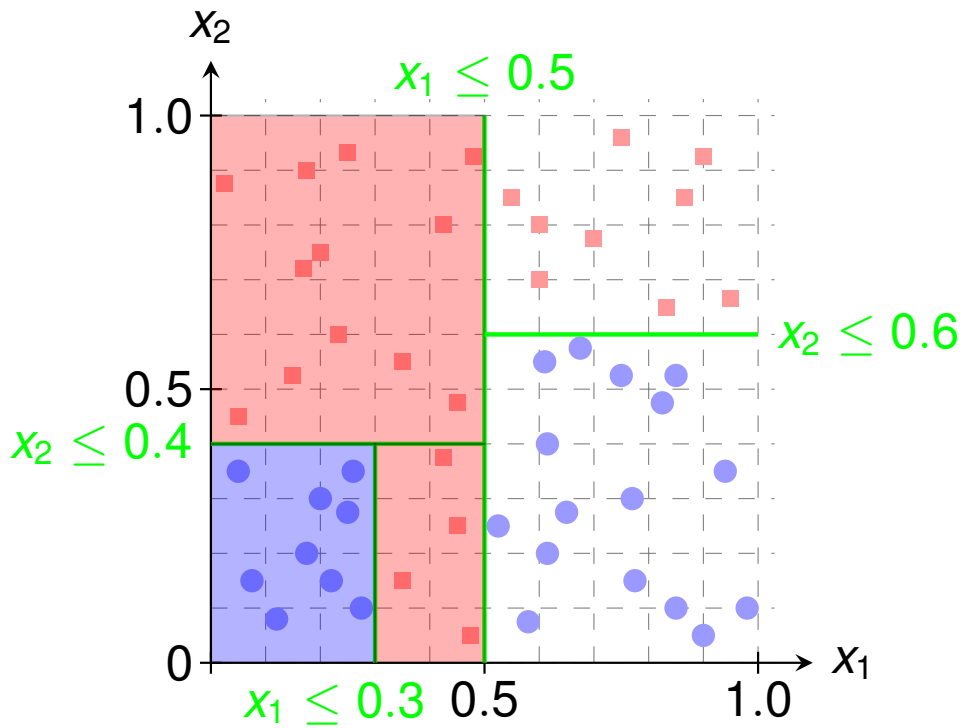
Intro: An Example of building a Decision Tree



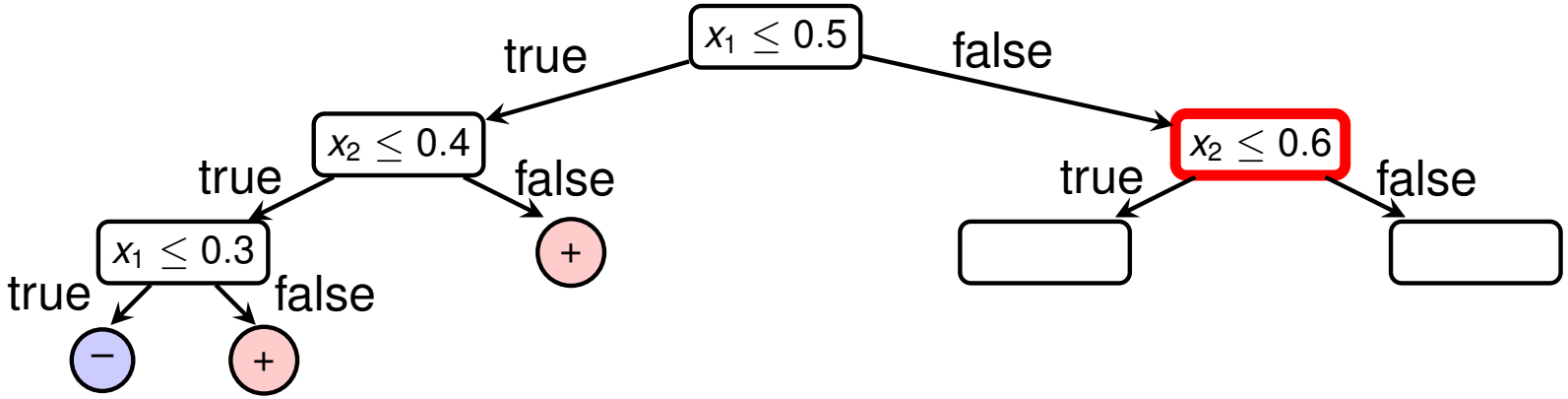


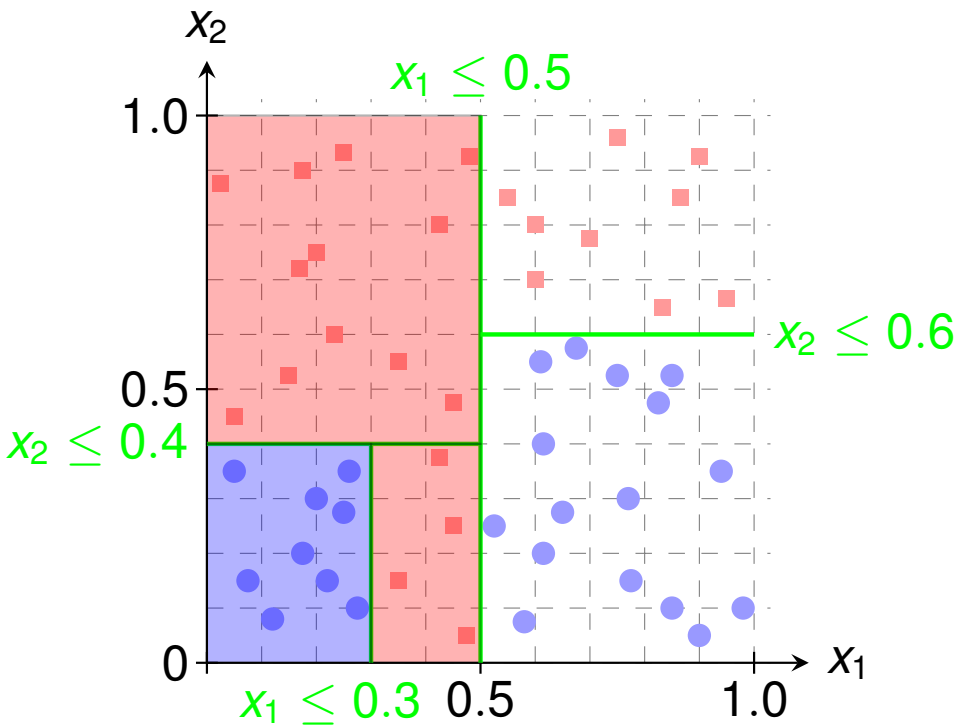
Intro: An Example of building a Decision Tree



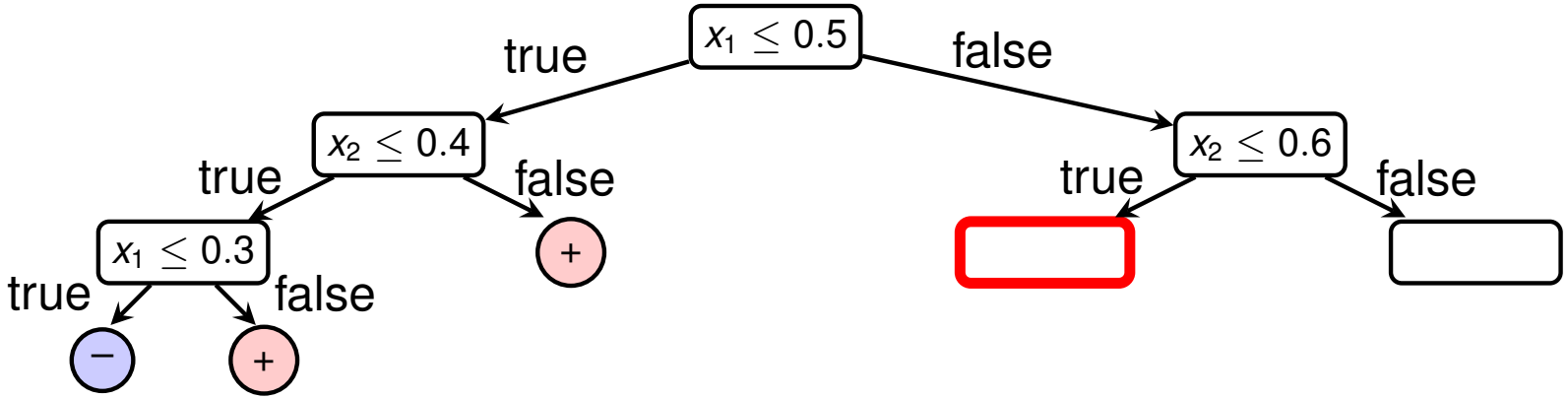


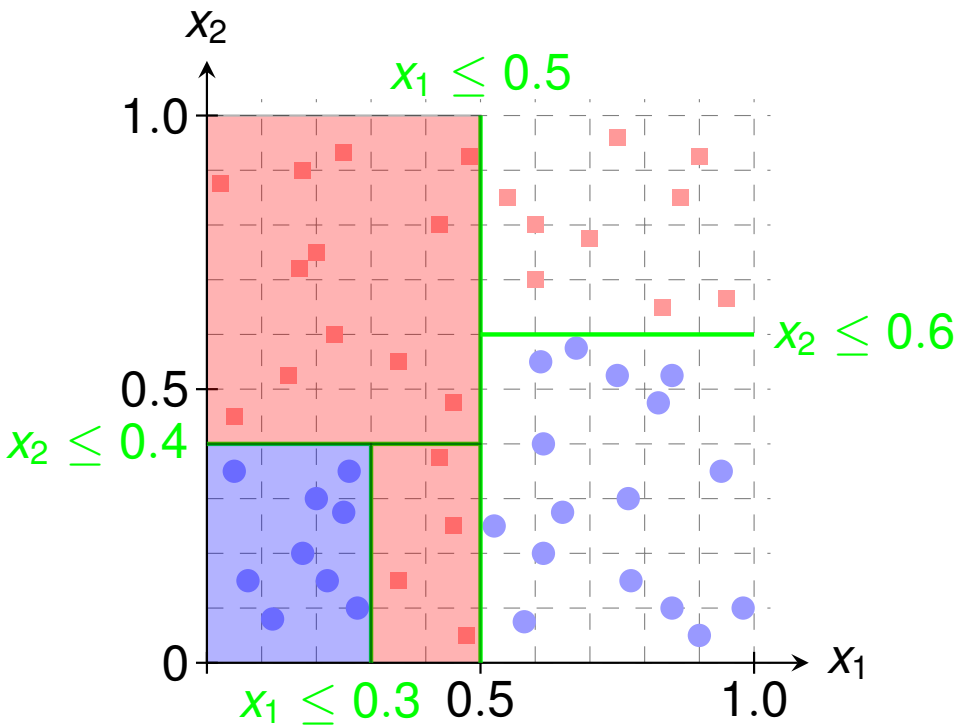
Intro: An Example of building a Decision Tree



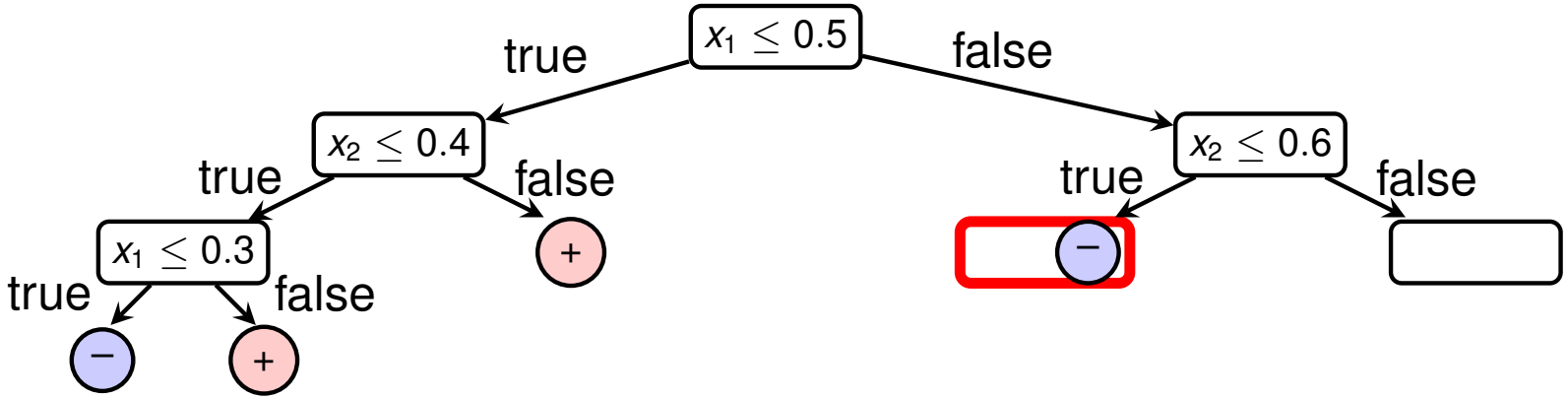


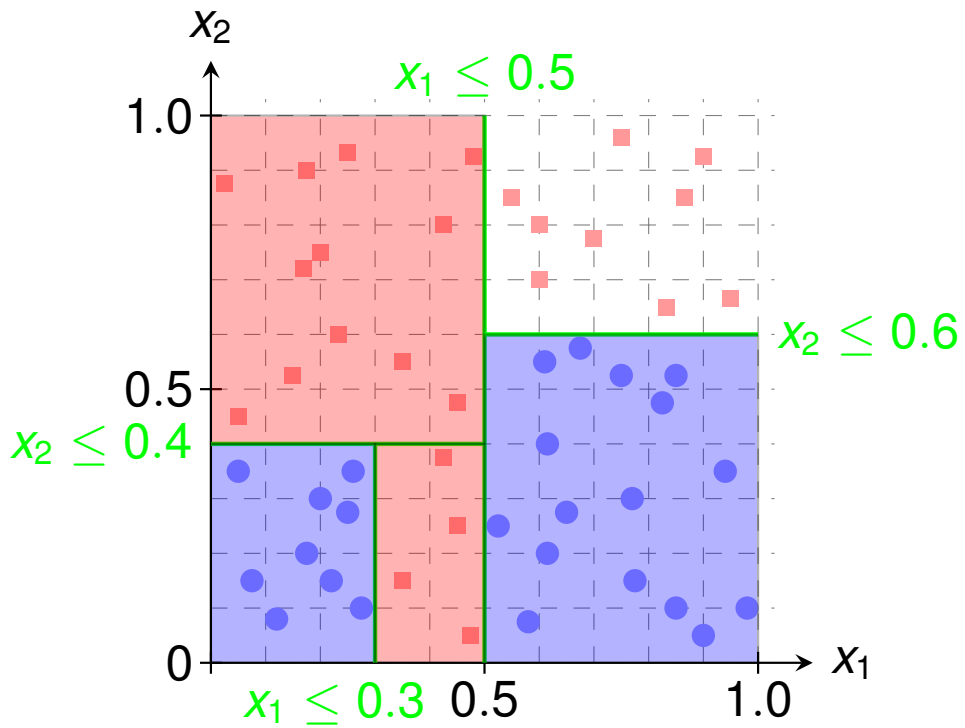
Intro: An Example of building a Decision Tree



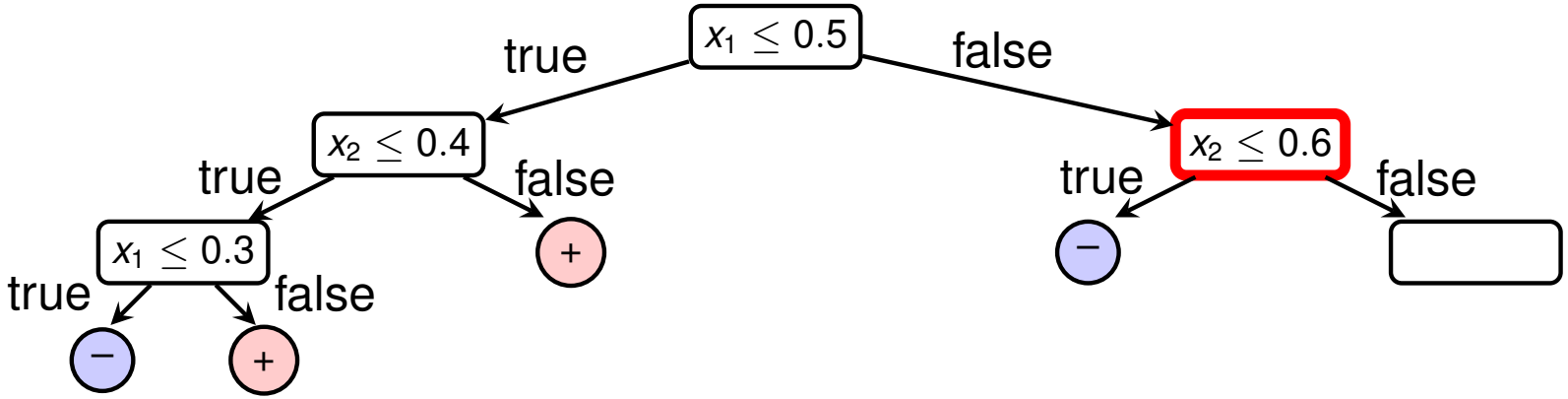


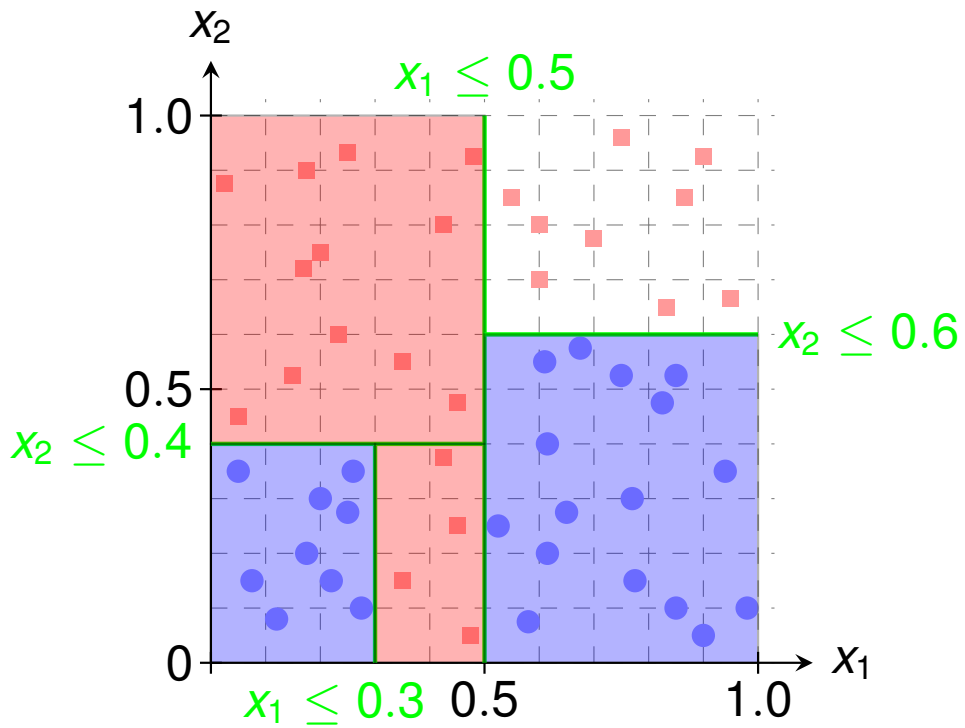
Intro: An Example of building a Decision Tree



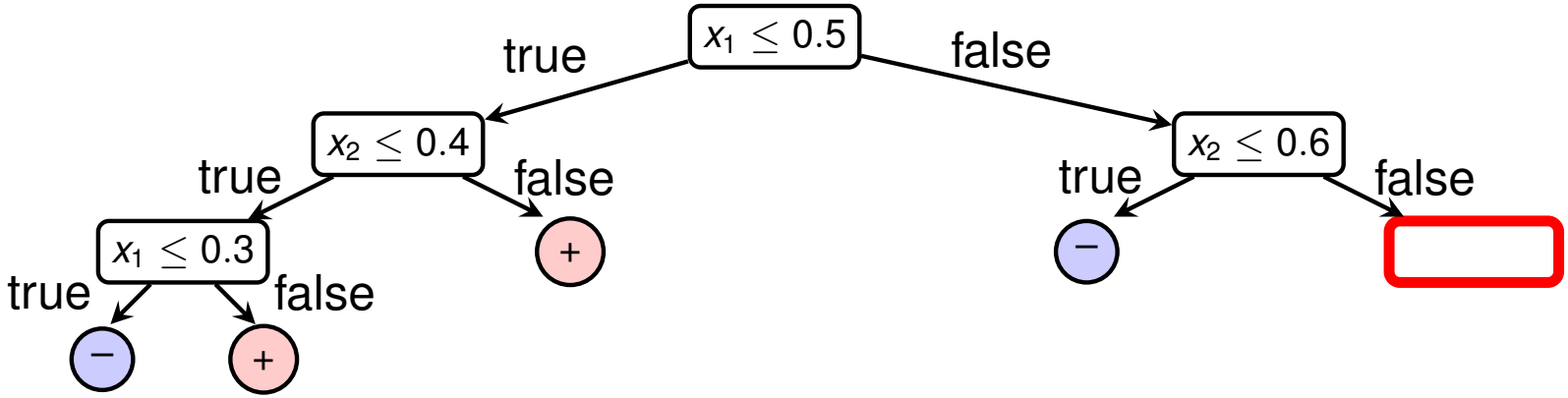


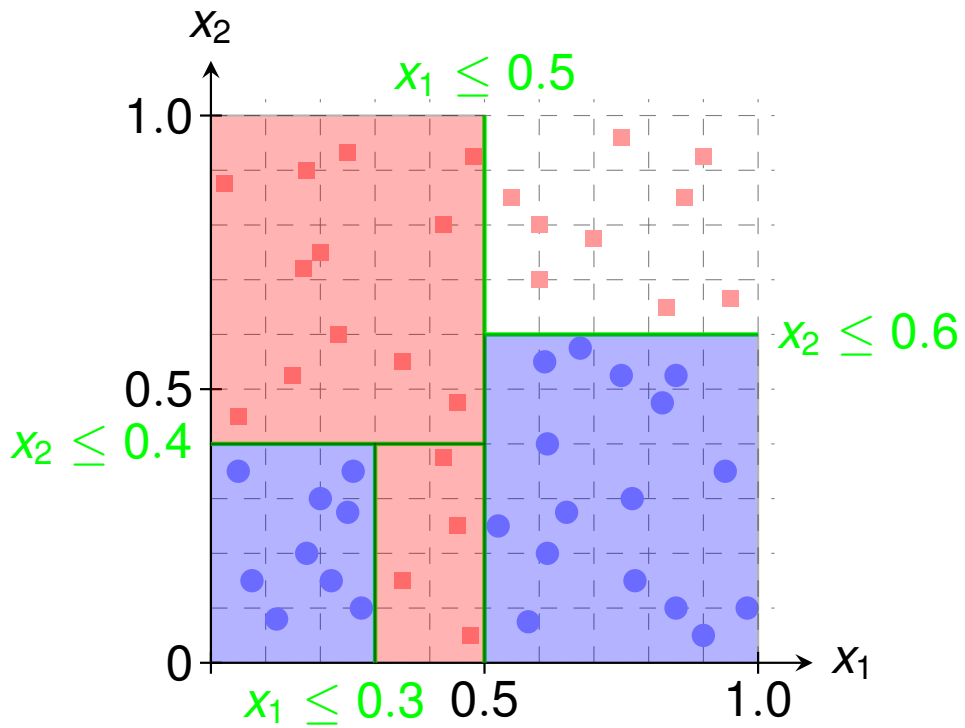
Intro: An Example of building a Decision Tree



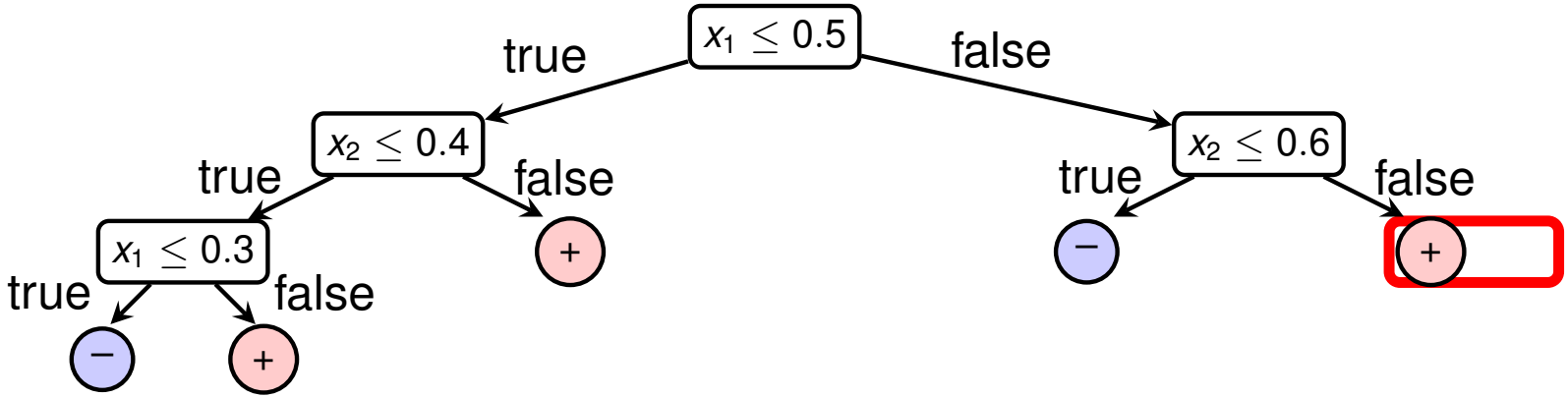


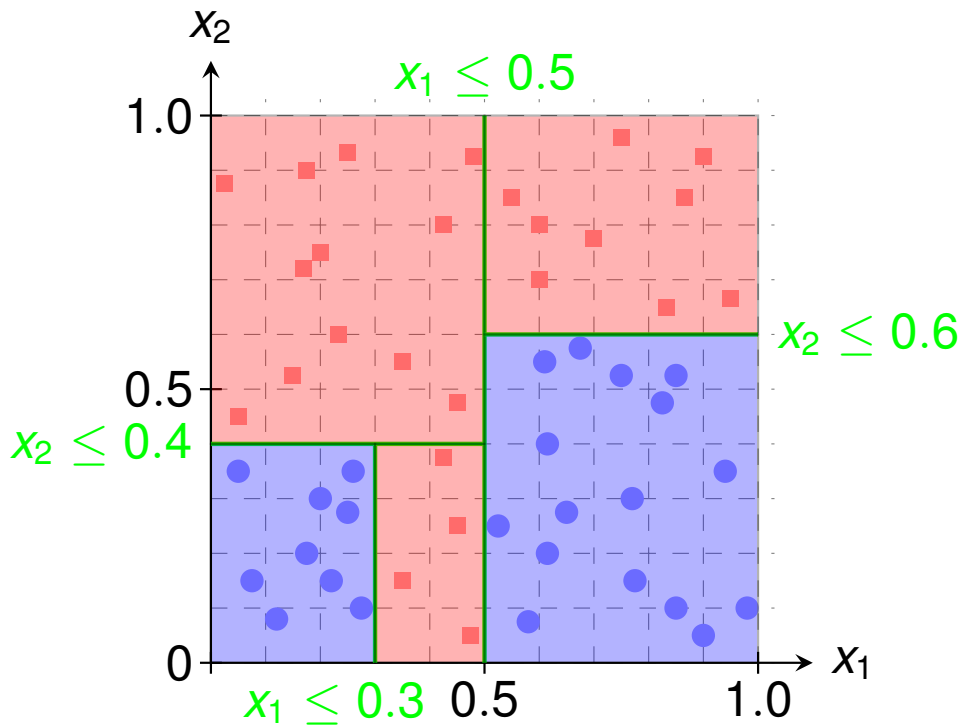
Intro: An Example of building a Decision Tree



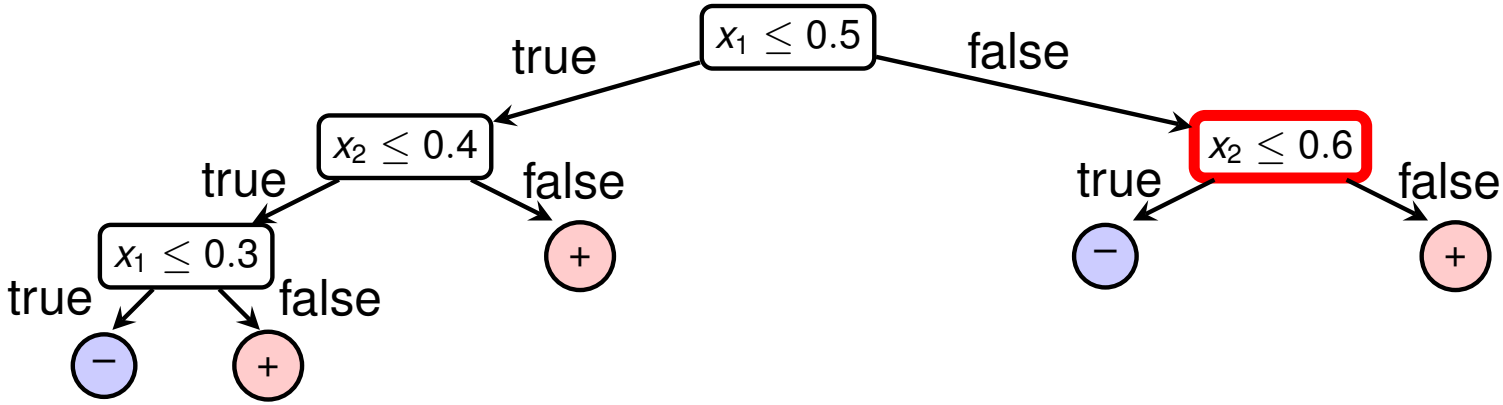


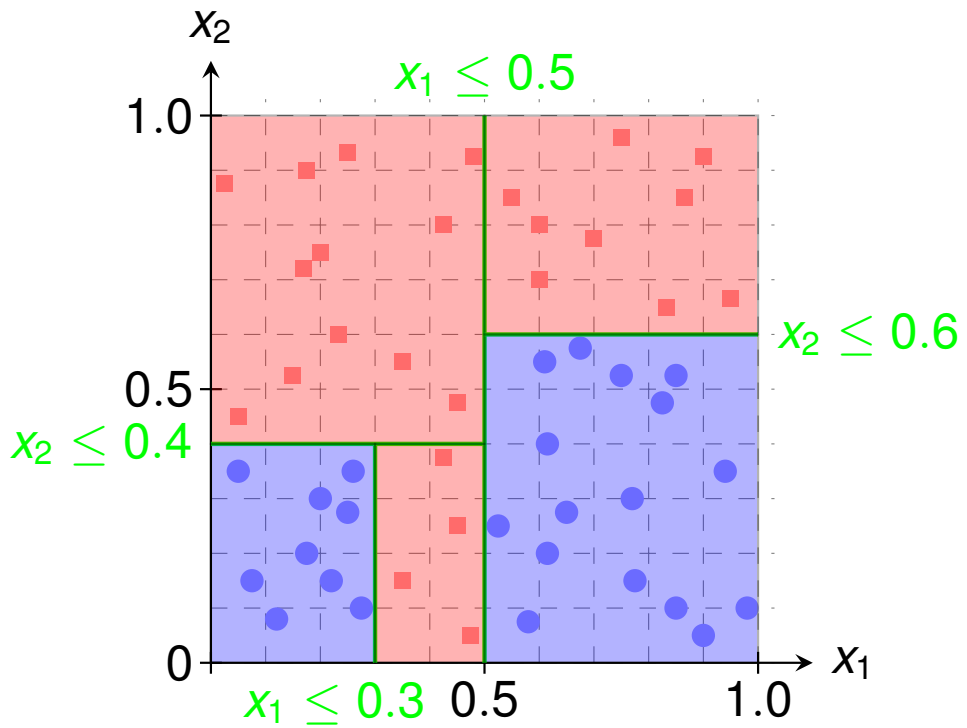
Intro: An Example of building a Decision Tree



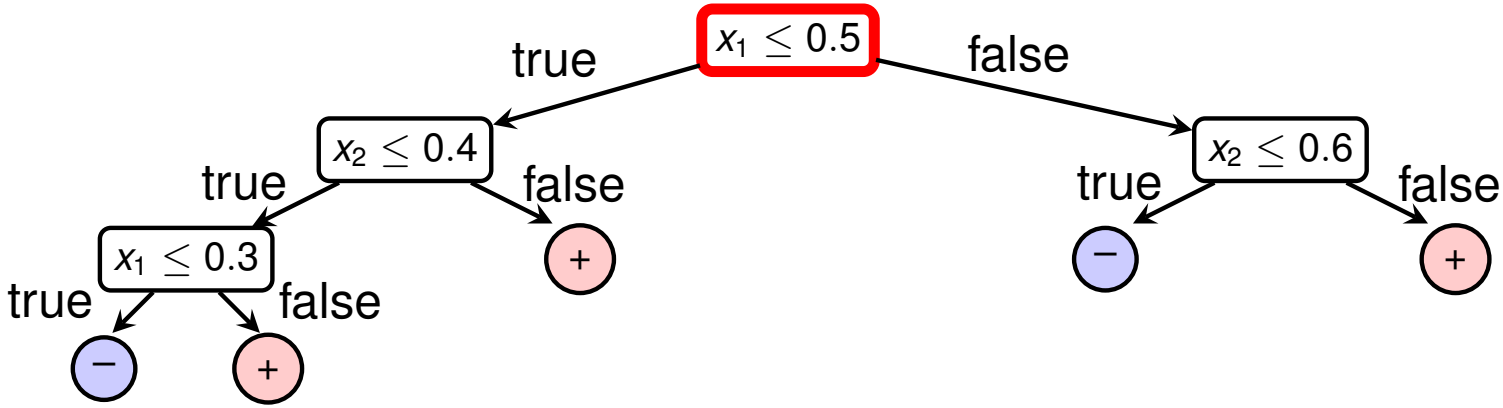


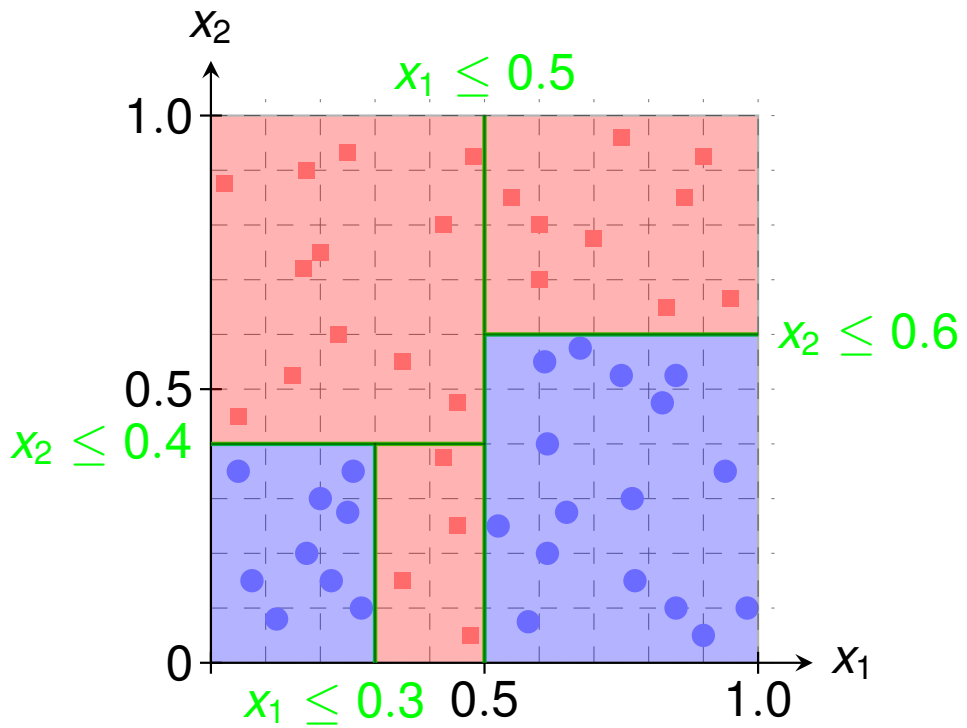
Intro: An Example of building a Decision Tree



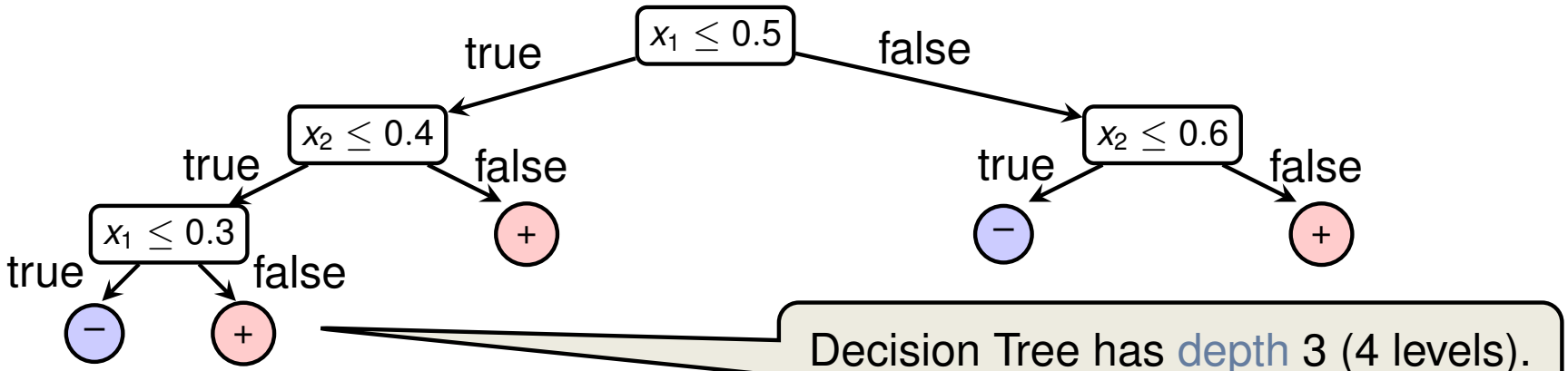


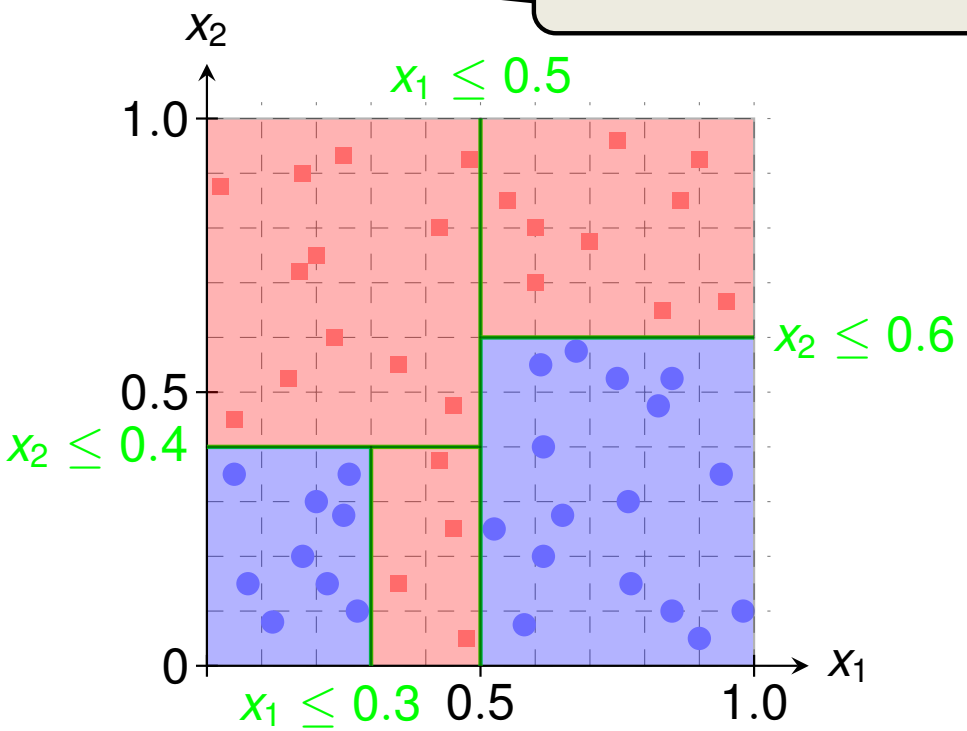
Intro: An Example of building a Decision Tree





Intro: An Example of building a Decision Tree



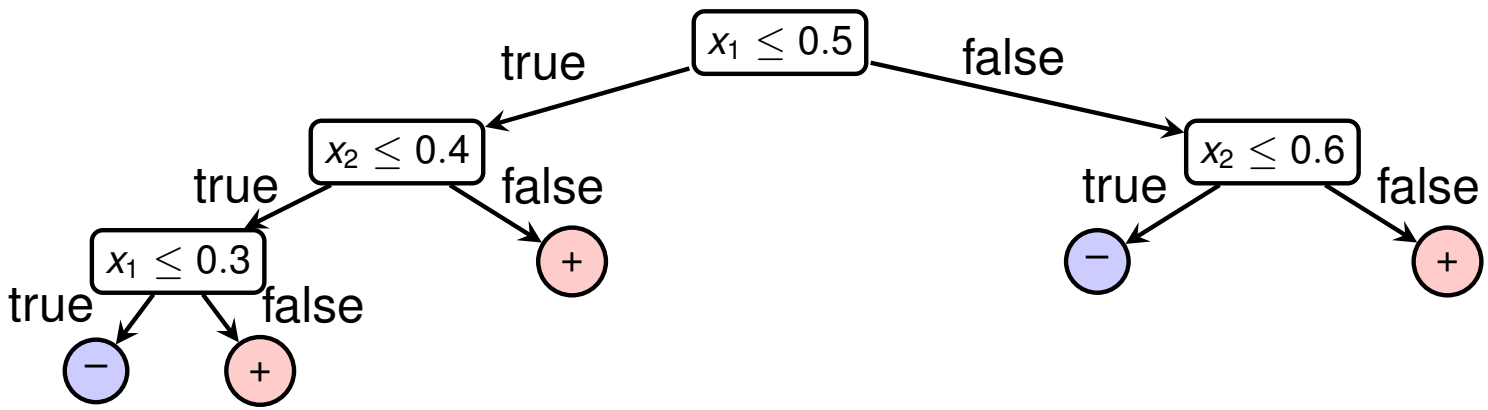


Making Predictions

Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf

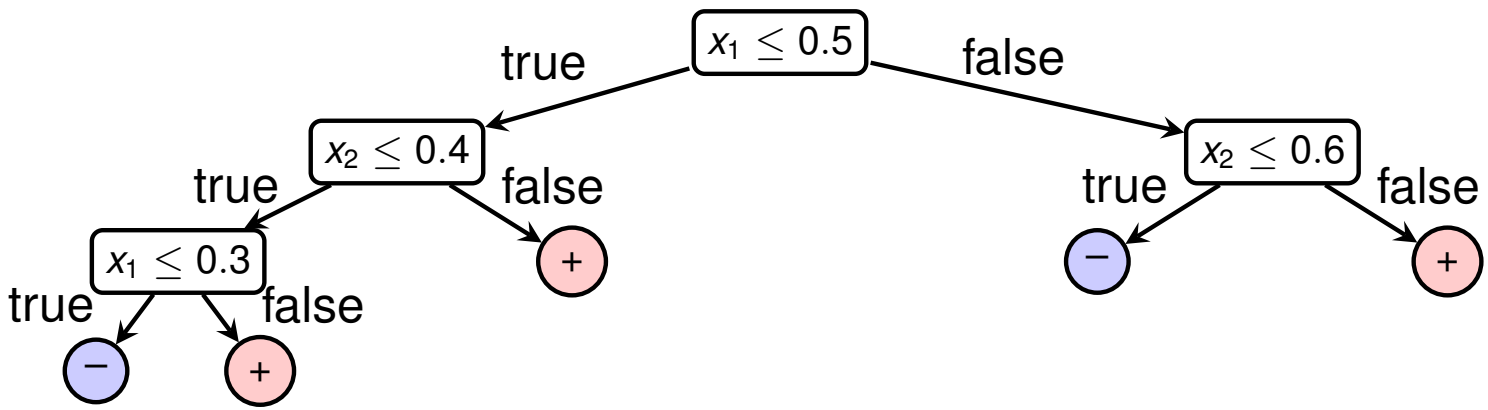
Making Predictions

Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



Making Predictions

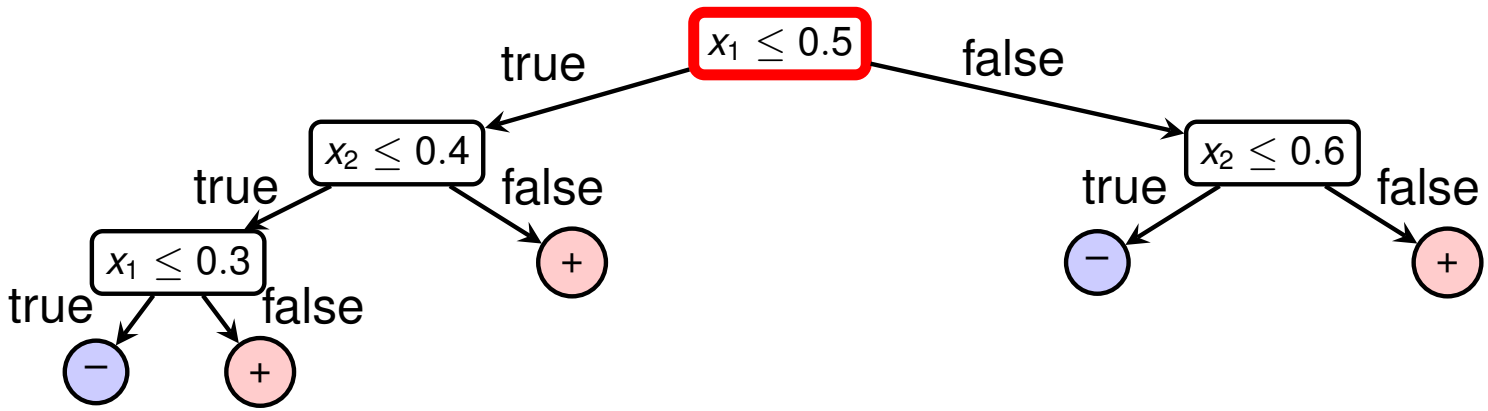
Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



Quiz 1: Classify point $(0.62, 0.53)$

Making Predictions

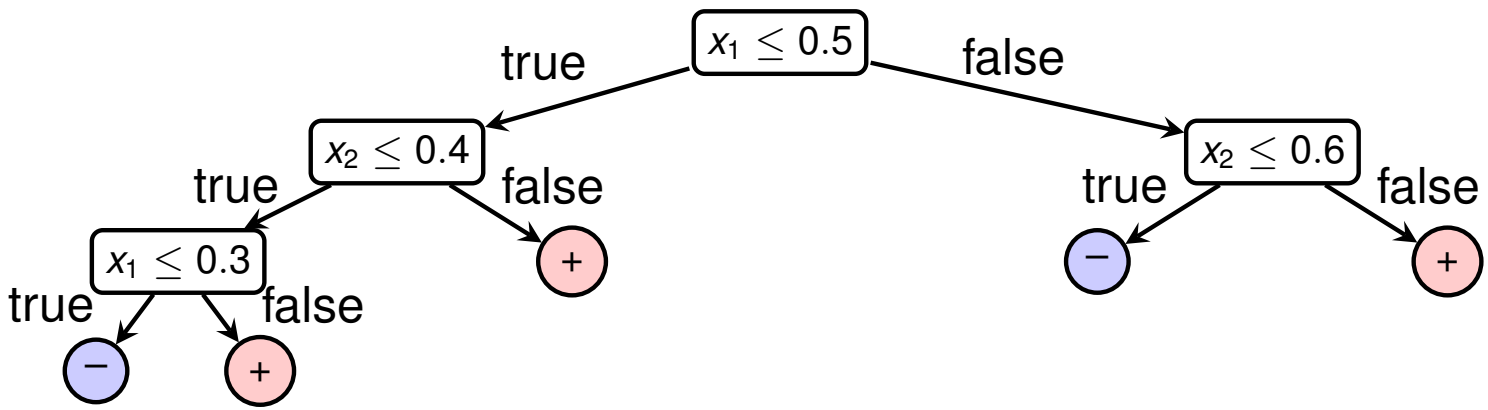
Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



Quiz 1: Classify point $(0.62, 0.53)$

Making Predictions

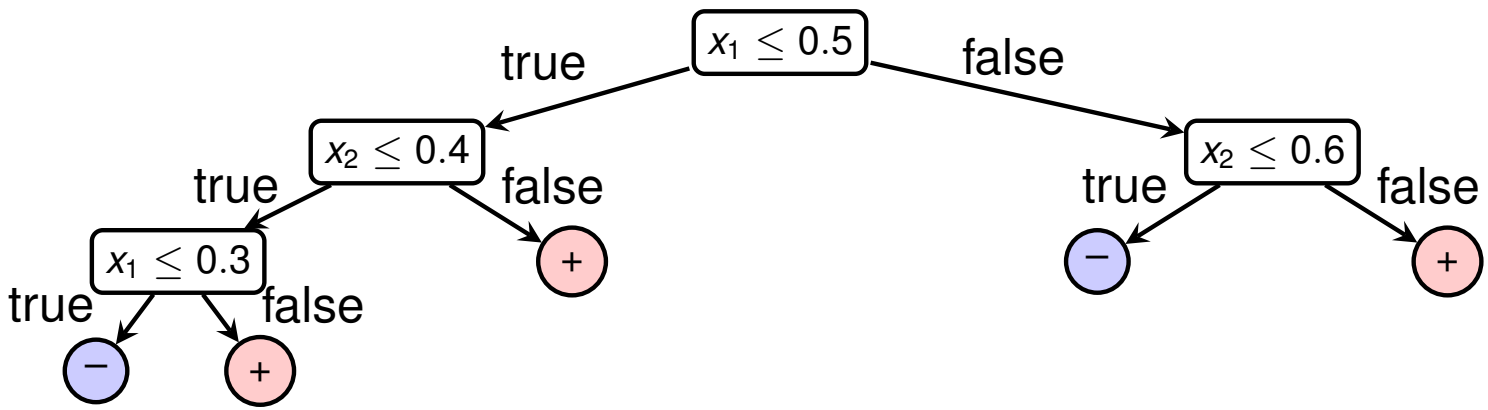
Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



Quiz 1: Classify point $(0.62, 0.53)$

Making Predictions

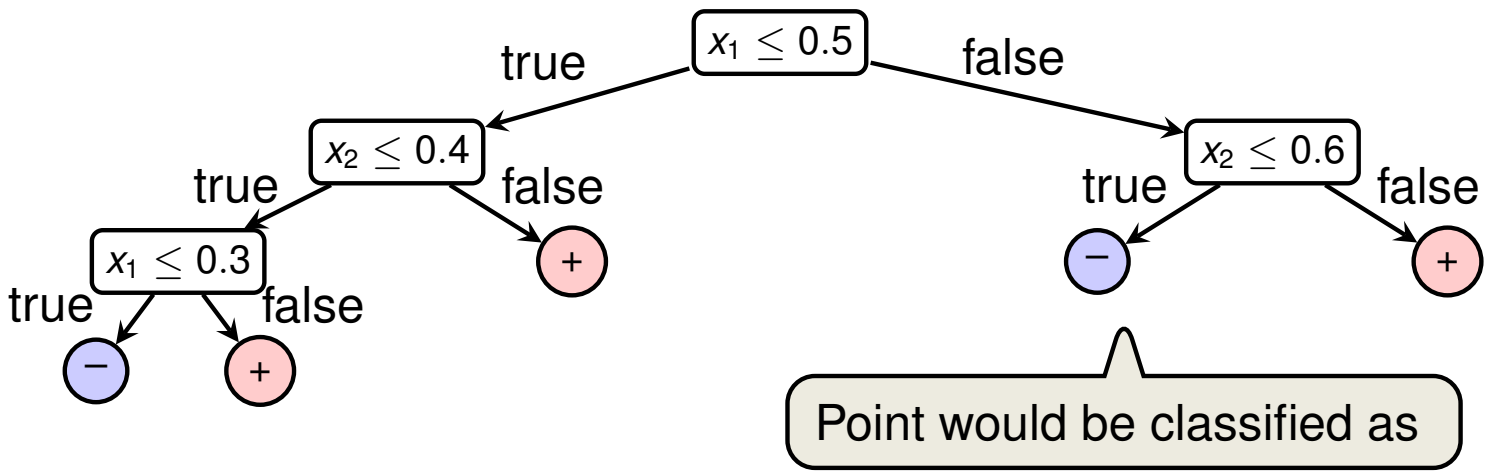
Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



Quiz 1: Classify point $(0.62, 0.53)$

Making Predictions

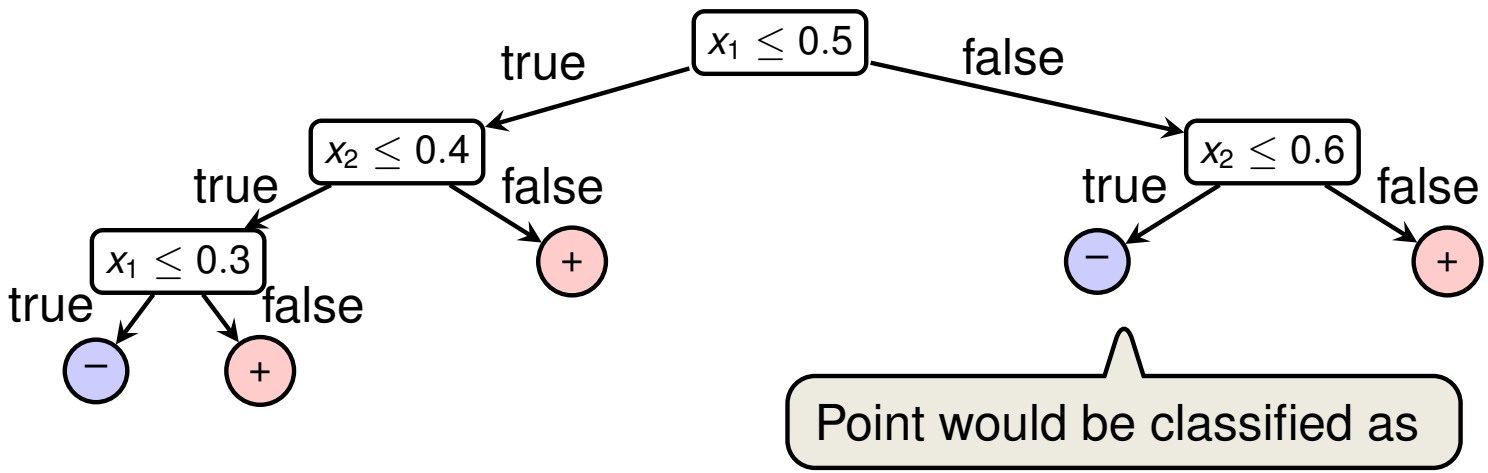
Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



Quiz 1: Classify point $(0.62, 0.53)$

Making Predictions

Classification: Given a new, unclassified example, follow a path down from the root until you hit a leaf



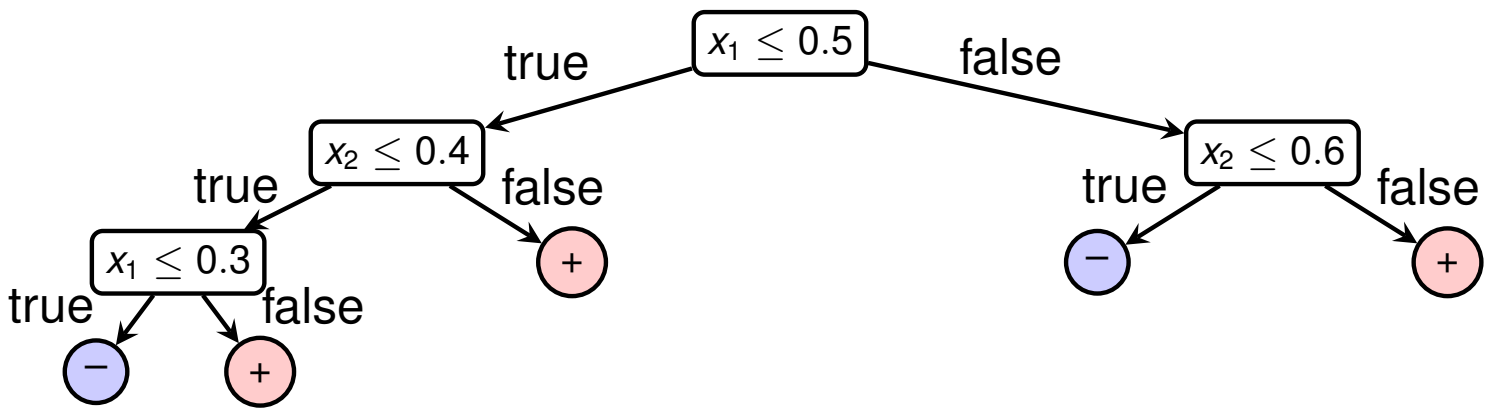
Quiz 1: Classify point $(0.62, 0.53)$

Challenge: Classification is easy once the decision tree is built, but how do we construct such a tree?

How to build a Decision Tree

Basic Idea

- Use a recursive approach, starting from the root which contains all data points in the training set
- After performing a split, the number of examples in each of descendant will be less
- Repeat splitting recursively unless you reach a node where all examples are of the same type
  $\Rightarrow$ node can be declared a leaf with (identical) label of the examples



The Iterative Dichotomizer 3-Algorithm

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathsf{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S\colon i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S\colon i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\mathsf{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathsf{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S\colon i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S\colon i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\mathsf{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathsf{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S\colon i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S\colon i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

Issues

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\mathrm{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathrm{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S\colon i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S\colon i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

Issues

- How do we define $\mathrm{Gain}(S, i)$ which decides the splitting?

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\mathrm{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathrm{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S\colon i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S\colon i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

Issues

- How do we define $\mathrm{Gain}(S, i)$ which decides the splitting?
- Even with a good splitting criterion, tree may become too large.

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\mathrm{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathrm{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S\colon i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S\colon i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

Issues

- How do we define $\mathrm{Gain}(S, i)$ which decides the splitting?
- Even with a good splitting criterion, tree may become too large.
  - Stopping Criterion: Stop splitting at a certain depth or when number of examples are below a certain threshold and take majority.

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\mathrm{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\mathrm{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S : i$ satisfies condition $C\}$)
6: Let the right subtree be the result of ID3($\{i \in S : i$ fails condition $C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

If classification is numerical, compute average or perform regression.

Issues

- How do we define $\mathrm{Gain}(S, i)$ which decides the splitting?
- Even with a good splitting criterion, tree may become too large.
  - Stopping Criterion: Stop splitting at a certain depth or when number of examples are below a certain threshold and take majority.

The Iterative Dichotomizer 3-Algorithm

Intuitively, $\text{Gain}(S, C)$ decribes the gain of information by knowing whether condition C holds for examples in S.

ID3 Algorithm

1: Let S be a set of labelled data points i
2: **If** all $y_i = 1$ **then** return a leaf with predictor 1
3: **If** all $y_i = 0$ **then** return a leaf with predictor 0
4: Let C be a condition on one feature that maximises $\text{Gain}(S, C)$
(If there is no such condition, return majority vote)
5: Let the left subtree be the result of ID3($\{i \in S : i \text{ satisfies condition } C\}$)
6: Let the right subtree be the result of ID3($\{i \in S : i \text{ fails condition } C\}$)
7: **return** tree that consists of a splitting node connected to both subtrees

If classification is numerical, compute average or perform regression.

Issues

- How do we define $\text{Gain}(S, i)$ which decides the splitting?
- Even with a good splitting criterion, tree may become too large.
  - Stopping Criterion: Stop splitting at a certain depth or when number of examples are below a certain threshold and take majority.
  - Pruning: After creating tree with ID3, check for the best non-leaf that can be made a leaf without increasing training error too much

Outline

Implementation of the Gain Measure

Implementation of the Gain Measure

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)

Implementation of the Gain Measure

Other popular gain measures are Information Gain, Gini Index or Entropy.

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)

Implementation of the Gain Measure

Other popular gain measures are Information Gain, Gini Index or Entropy.

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)
- Define $f(x) = \min\{x, 1-x\}$ for $x \in [0,1]$.

Implementation of the Gain Measure

Other popular gain measures are Information Gain, Gini Index or Entropy.

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)
- Define $f(x) = \min\{x, 1-x\}$ for $x \in [0,1]$.
- Let $f(\mathbb{P}_{i\sim S}[y_i = 1])$ be the training error before the splitting, where the probability is uniform over all examples in S
  (recall: if we don't split, we classify according to majority vote)

Implementation of the Gain Measure

Other popular gain measures are Information Gain, Gini Index or Entropy.

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)
- Define $f(x) = \min\{x, 1-x\}$ for $x \in [0,1]$.
- Let $f(\mathbb{P}_{i\sim S}[y_i = 1])$ be the training error before the splitting, where the probability is uniform over all examples in S
  (recall: if we don't split, we classify according to majority vote)
- **Exercise**: What is the training error after splitting according to condition C?

Implementation of the Gain Measure

Other popular gain measures are Information Gain, Gini Index or Entropy.

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)
- Define $f(x) = \min\{x, 1-x\}$ for $x \in [0,1]$.
- Let $f(\mathbb{P}_{i\sim S}[y_i = 1])$ be the training error before the splitting, where the probability is uniform over all examples in S
  (recall: if we don't split, we classify according to majority vote)
- **Exercise**: What is the training error after splitting according to condition C?

Solution to Exercise:

$$\mathbb{P}_{i\sim S}[i \text{ satisfies } C] \cdot f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) +$$
$$\mathbb{P}_{i\sim S}[i \text{ fails } C] \cdot f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C])$$

Implementation of the Gain Measure

Other popular gain measures are Information Gain, Gini Index or Entropy.

Training Error

- one of the simplest definition of gain (yet still a bit technical ☹)
- Define $f(x) = \min\{x, 1-x\}$ for $x \in [0,1]$.
- Let $f(\mathbb{P}_{i\sim S}[y_i = 1])$ be the training error before the splitting, where the probability is uniform over all examples in S
  (recall: if we don't split, we classify according to majority vote)
- **Exercise**: What is the training error after splitting according to condition C?
- $\text{Gain}(S, C)$ is then defined as the decrease in the training error

Solution to Exercise:

$$\mathbb{P}_{i\sim S}[i \text{ satisfies } C] \cdot f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) +$$
$$\mathbb{P}_{i\sim S}[i \text{ fails } C] \cdot f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C])$$

Illustration of Training Error as Gain Measure

- Consider first step of the algorithm

Illustration of Training Error as Gain Measure

- Consider first step of the algorithm

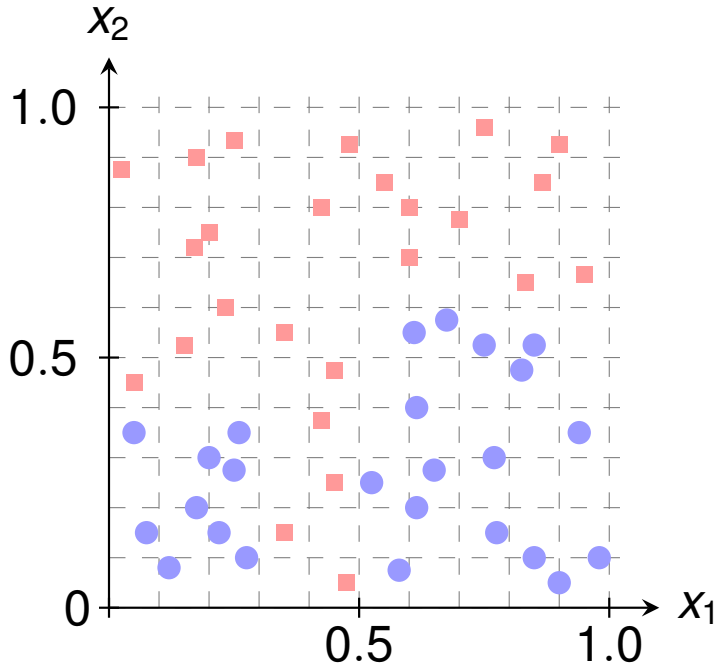


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50}$$

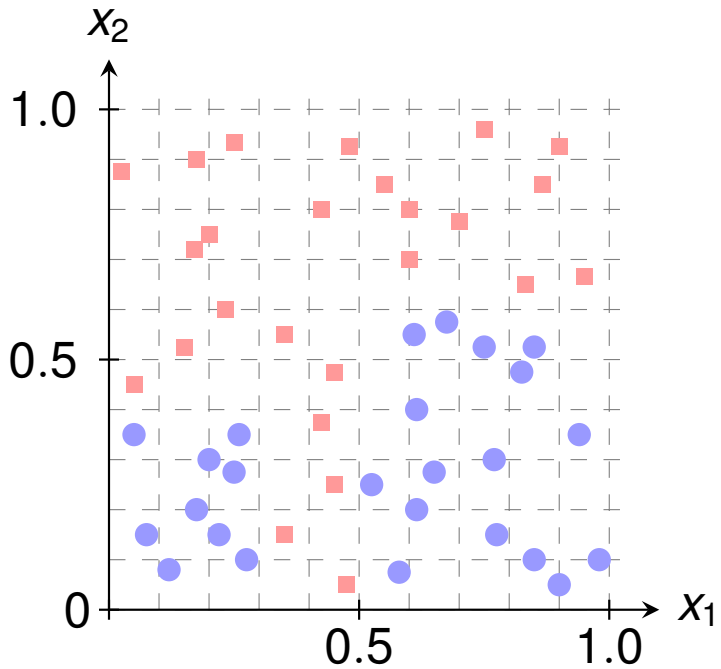


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

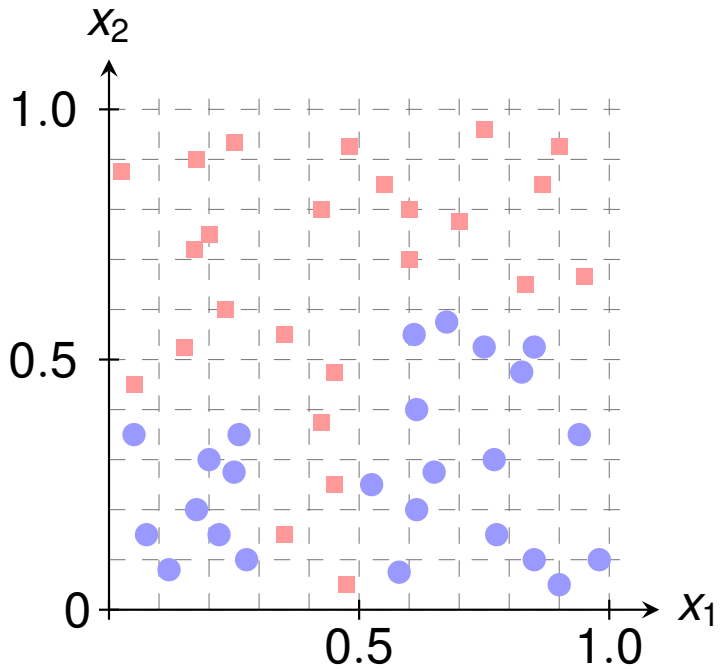


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_1 \leq 0.5$:

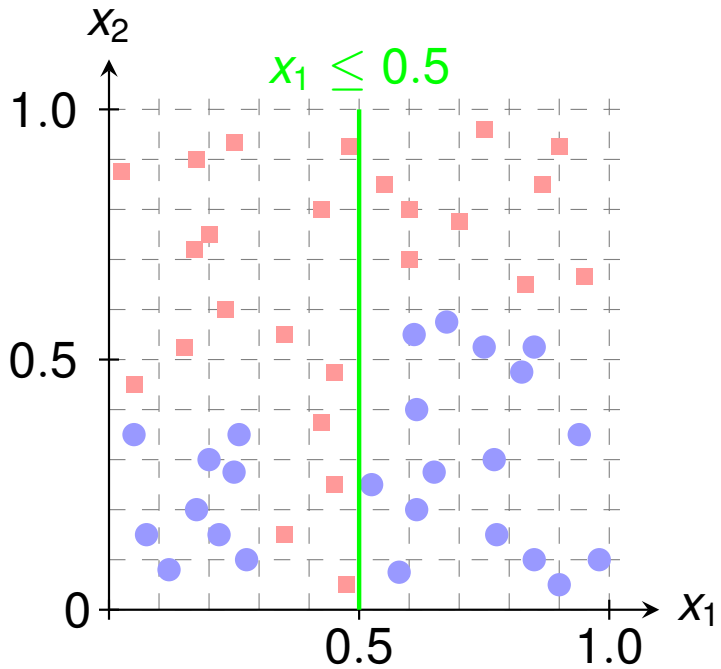


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_1 \leq 0.5$:
  - $x_1 \leq 0.5$: 9 blue points, 16 red points

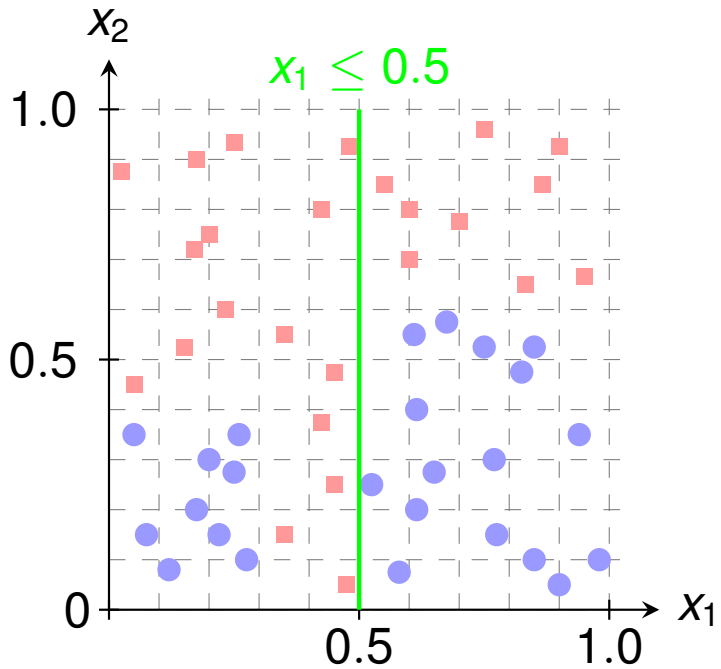


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_1 \leq 0.5$:
  - $x_1 \leq 0.5$: 9 blue points, 16 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{9}{25}$$

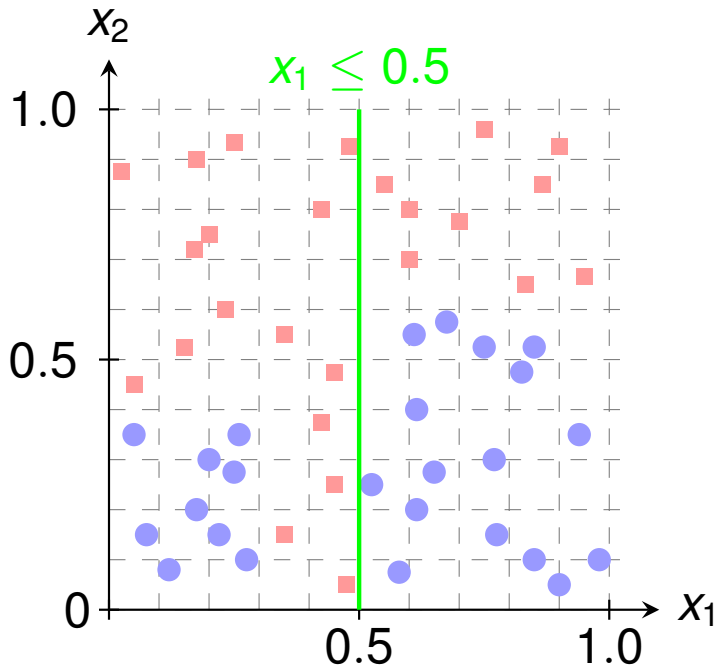


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_1 \leq 0.5$:
  - $x_1 \leq 0.5$: 9 blue points, 16 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{9}{25}$$

  - $x_1 > 0.5$: 16 blue points, 9 red points

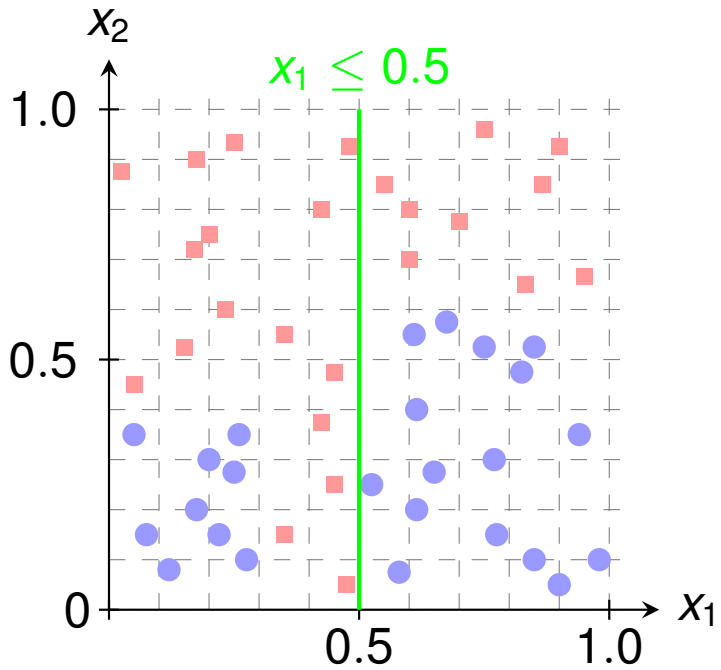


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_1 \leq 0.5$:
  - $x_1 \leq 0.5$: 9 blue points, 16 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{9}{25}$$

  - $x_1 > 0.5$: 16 blue points, 9 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C]) = \frac{9}{25}$$

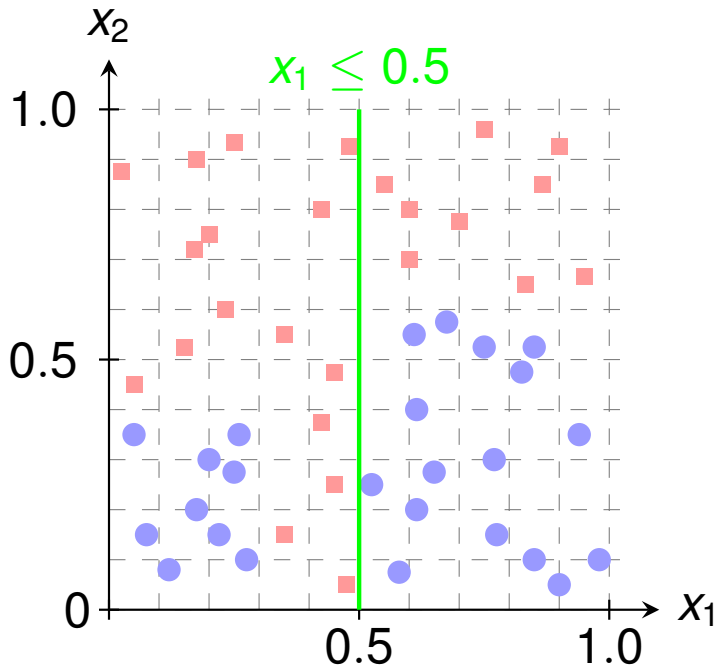


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_1 \leq 0.5$:
  - $x_1 \leq 0.5$: 9 blue points, 16 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{9}{25}$$

  - $x_1 > 0.5$: 16 blue points, 9 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C]) = \frac{9}{25}$$

  - Gain (decrease in training error) is:

$$0.5 - \frac{25}{50} \cdot \frac{9}{25} - \frac{25}{50} \cdot \frac{9}{25} = 0.14$$

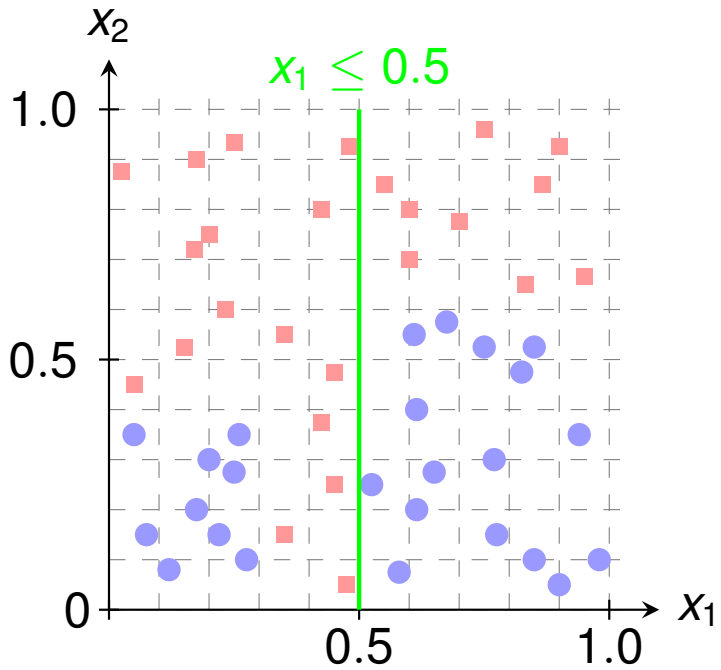


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:

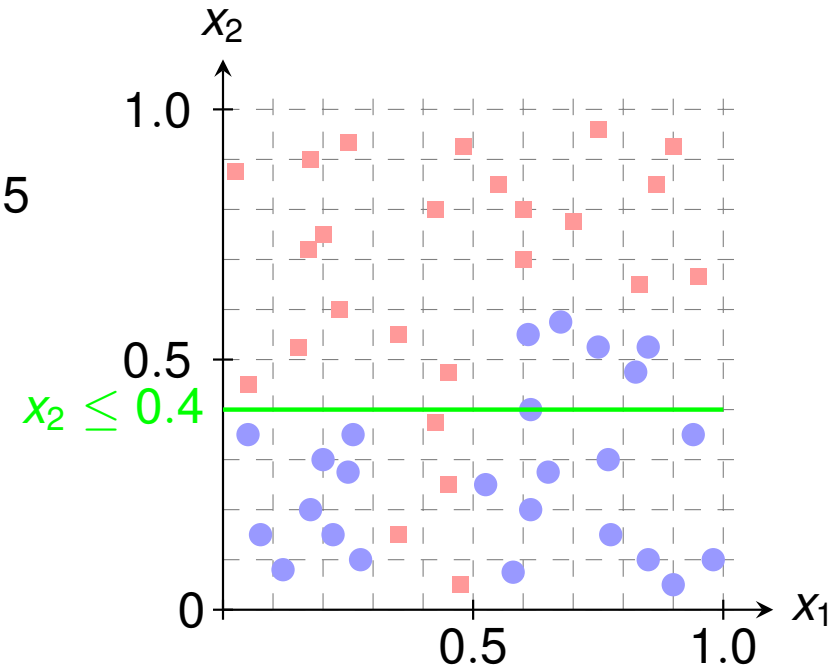


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:
  - $x_2 \leq 0.4$: 20 blue points, 4 red points

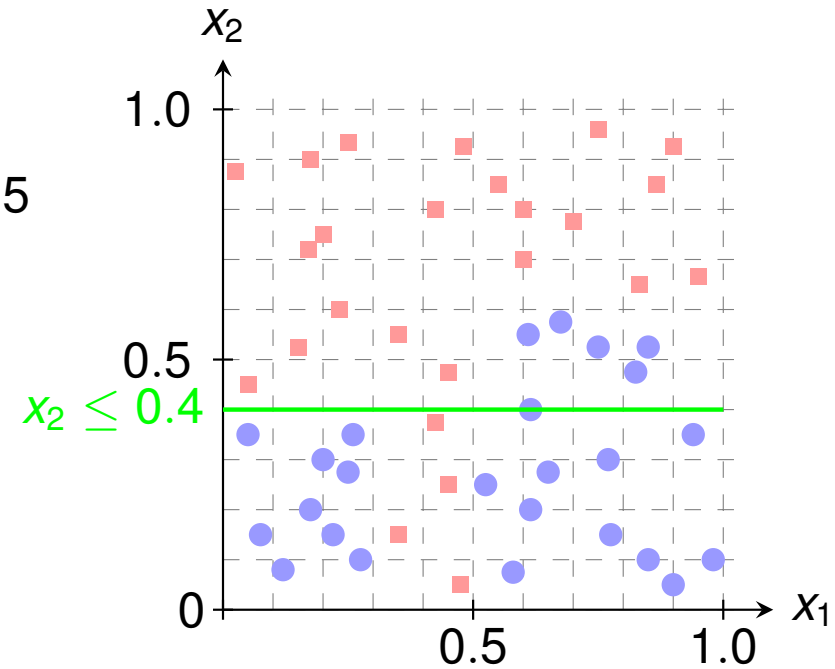


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:
  - $x_2 \leq 0.4$: 20 blue points, 4 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{4}{24}$$

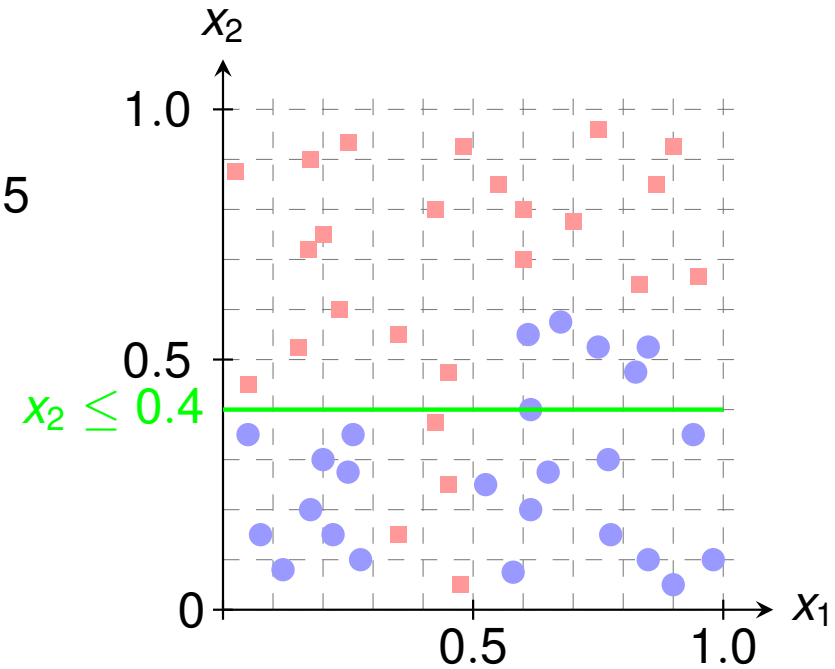


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:
  - $x_2 \leq 0.4$: 20 blue points, 4 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{4}{24}$$

  - $x_2 > 0.4$: 5 blue points, 21 red points

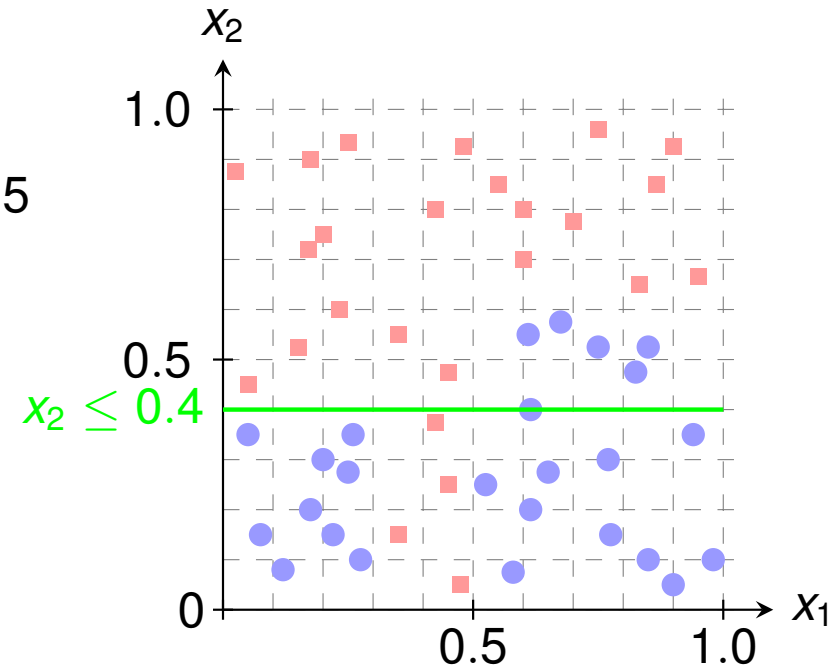


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:
  - $x_2 \leq 0.4$: 20 blue points, 4 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{4}{24}$$

  - $x_2 > 0.4$: 5 blue points, 21 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C]) = \frac{5}{26}$$

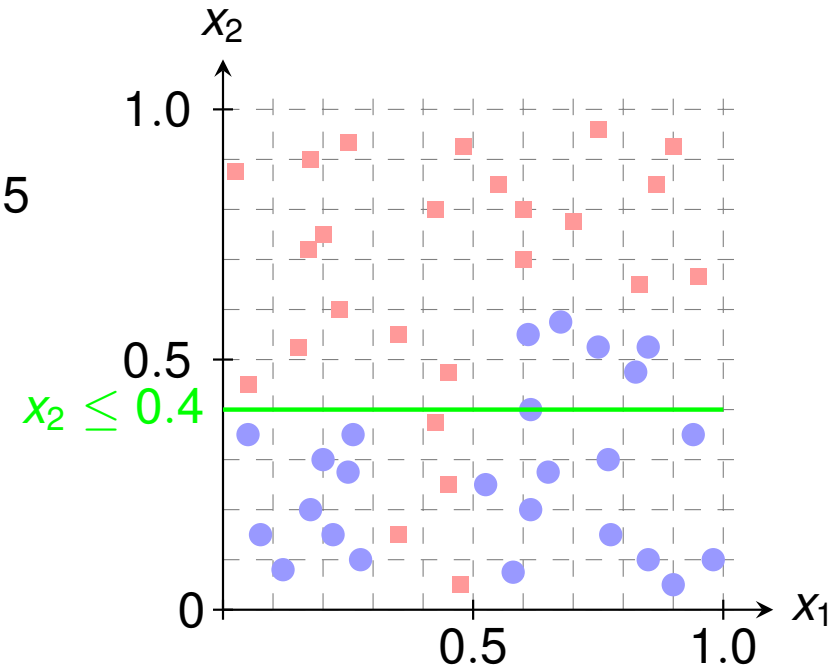


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:
  - $x_2 \leq 0.4$: 20 blue points, 4 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{4}{24}$$

  - $x_2 > 0.4$: 5 blue points, 21 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C]) = \frac{5}{26}$$

  - Gain (decrease in training error) is:

$$0.5 - \frac{24}{50} \cdot \frac{4}{24} - \frac{26}{50} \cdot \frac{5}{26} = 0.32$$

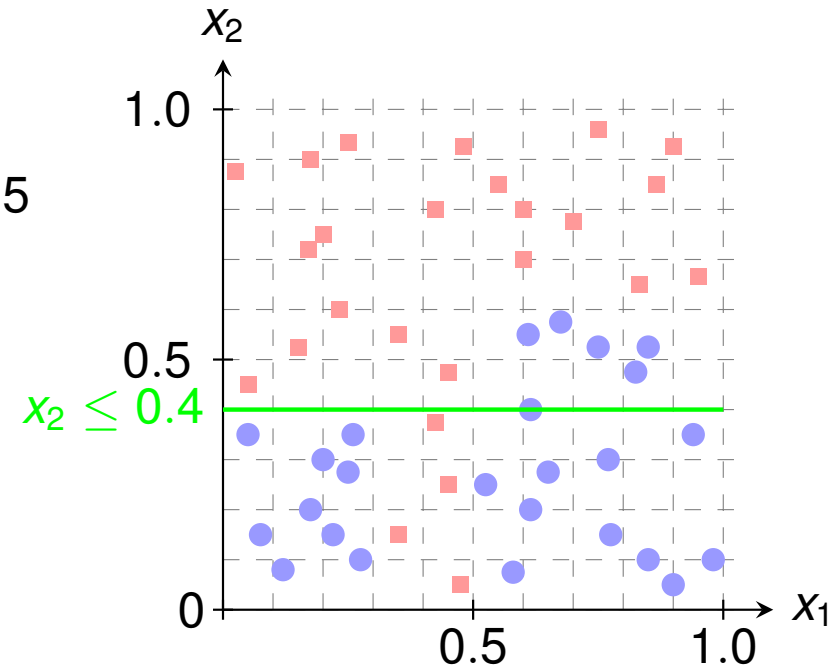


Illustration of Training Error as Gain Measure

- Consider first step of the algorithm
- Before splitting, training error is:

$$\mathbb{P}_{i\sim S}[y_i = 1] = \frac{25}{50} \Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1])) = 0.5$$

- Consider the split $x_2 \leq 0.4$:
  - $x_2 \leq 0.4$: 20 blue points, 4 red points

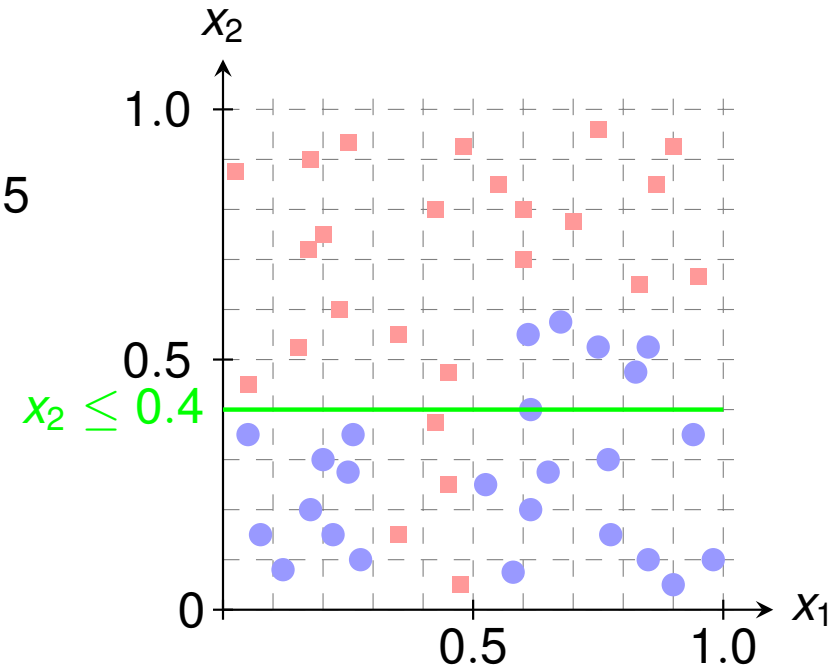
$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ satisfies } C]) = \frac{4}{24}$$

  - $x_2 > 0.4$: 5 blue points, 21 red points

$$\Rightarrow f(\mathbb{P}_{i\sim S}[y_i = 1 \mid i \text{ fails } C]) = \frac{5}{26}$$

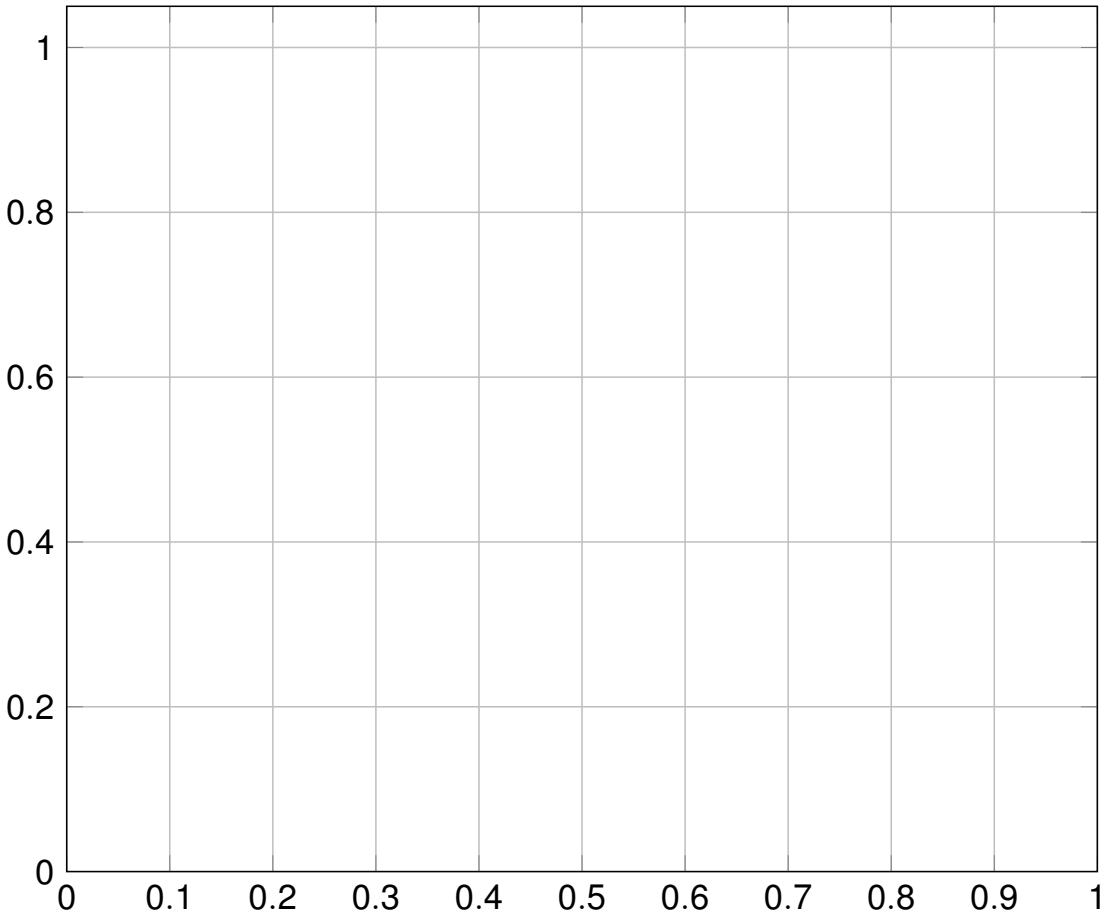
  - Gain (decrease in training error) is:

$$0.5 - \frac{24}{50} \cdot \frac{4}{24} - \frac{26}{50} \cdot \frac{5}{26} = 0.32$$



The split $x_2 \leq 0.4$ achieves a much larger gain than $x_1 \leq 0.5$ (which was done in the initial example!)

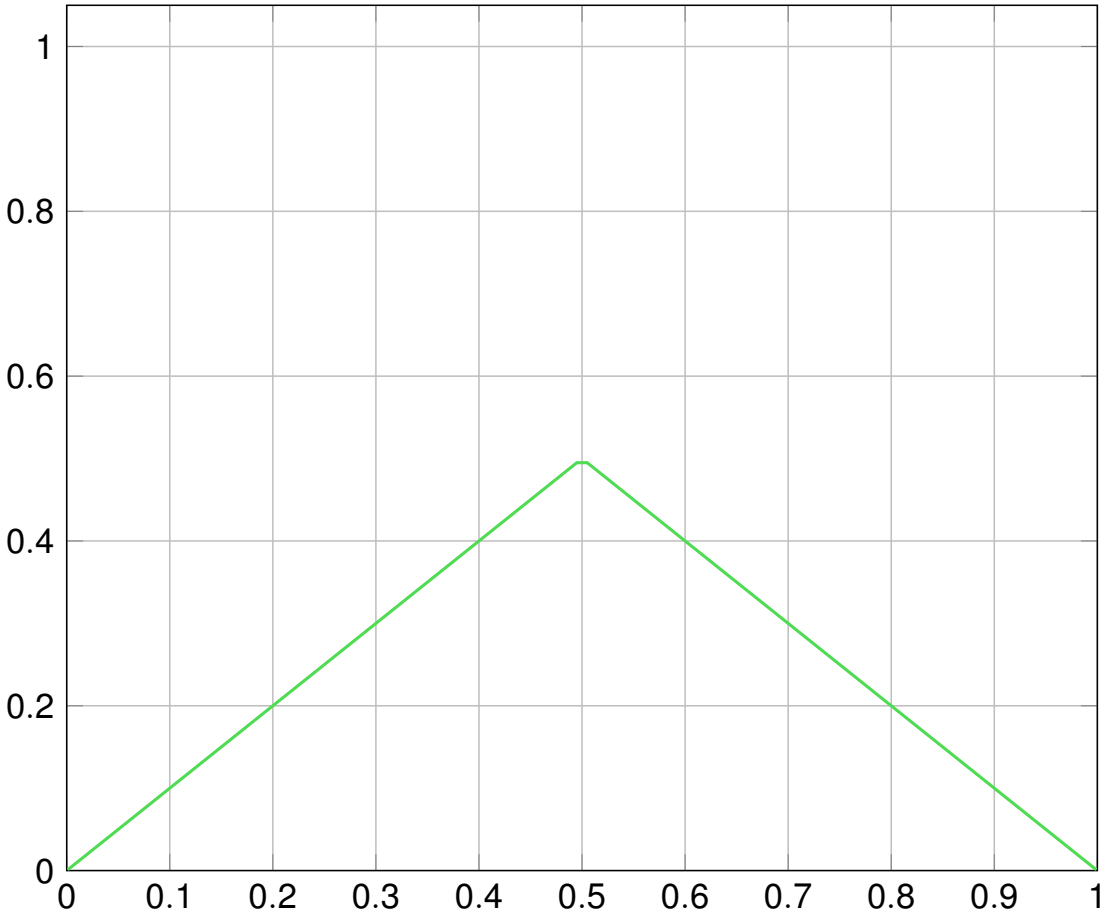
Other Gain Measures



Other Gain Measures

Let $x \in [0, 1]$ be the fraction of positive training examples.

Other Gain Measures

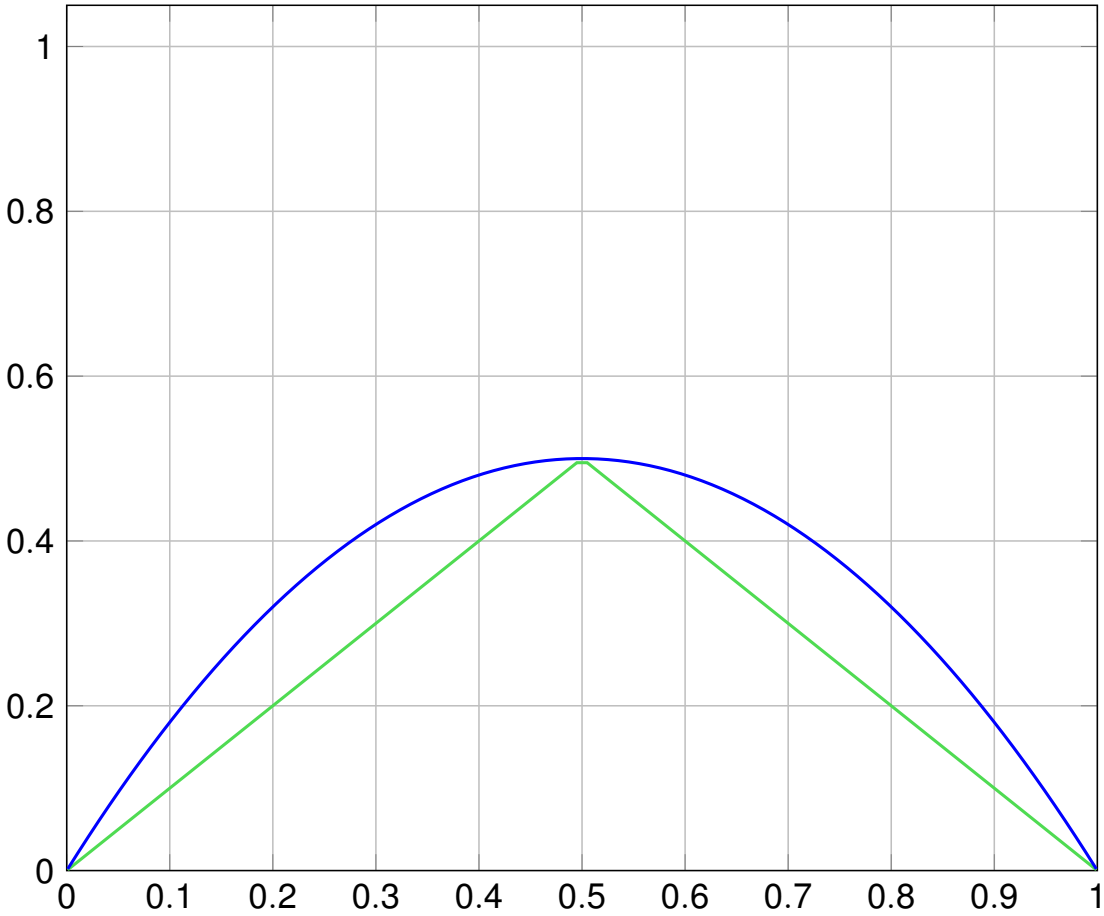


Other Gain Measures

Let $x \in [0, 1]$ be the fraction of positive training examples.

- Training Error: $\min(x, 1 - x)$

Other Gain Measures

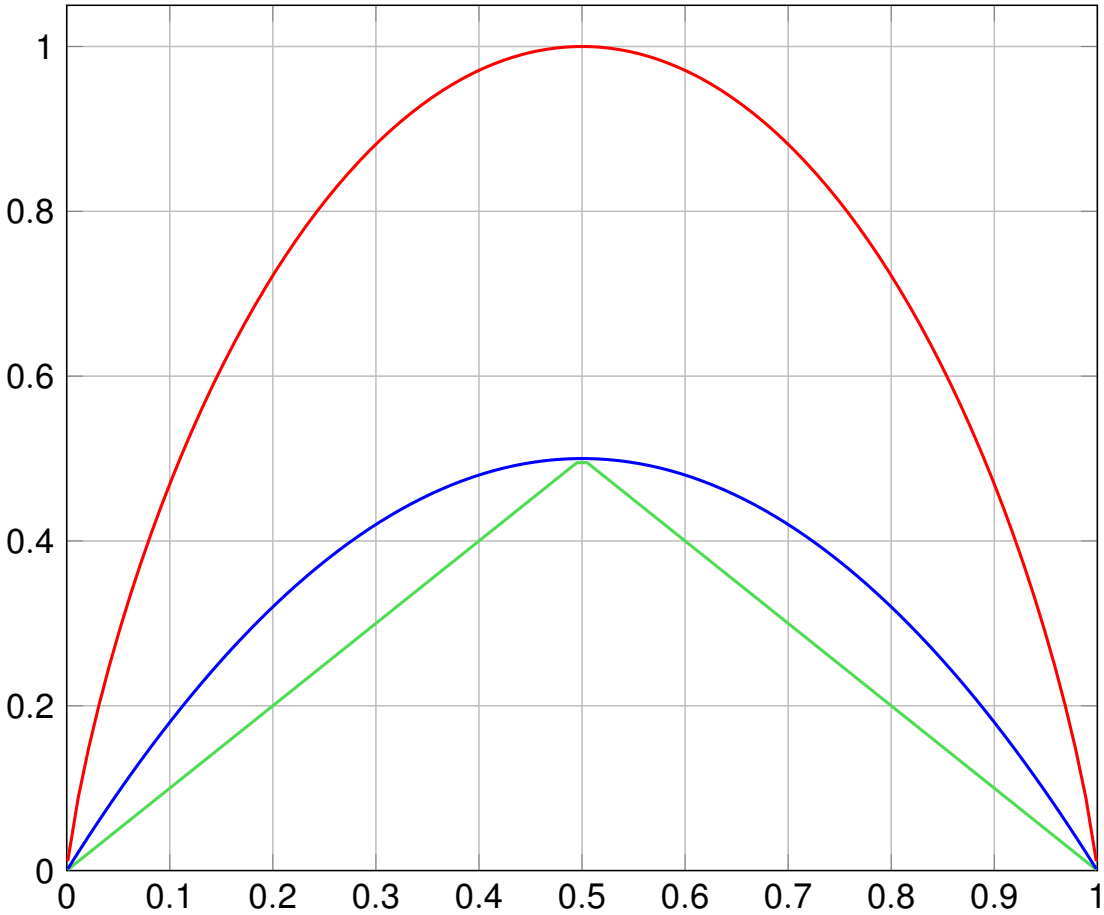


Other Gain Measures

Let $x \in [0, 1]$ be the fraction of positive training examples.

- Training Error: $\min(x, 1 - x)$
- Gini Impurity: $x(1 - x) + (1 - x)x$

Other Gain Measures

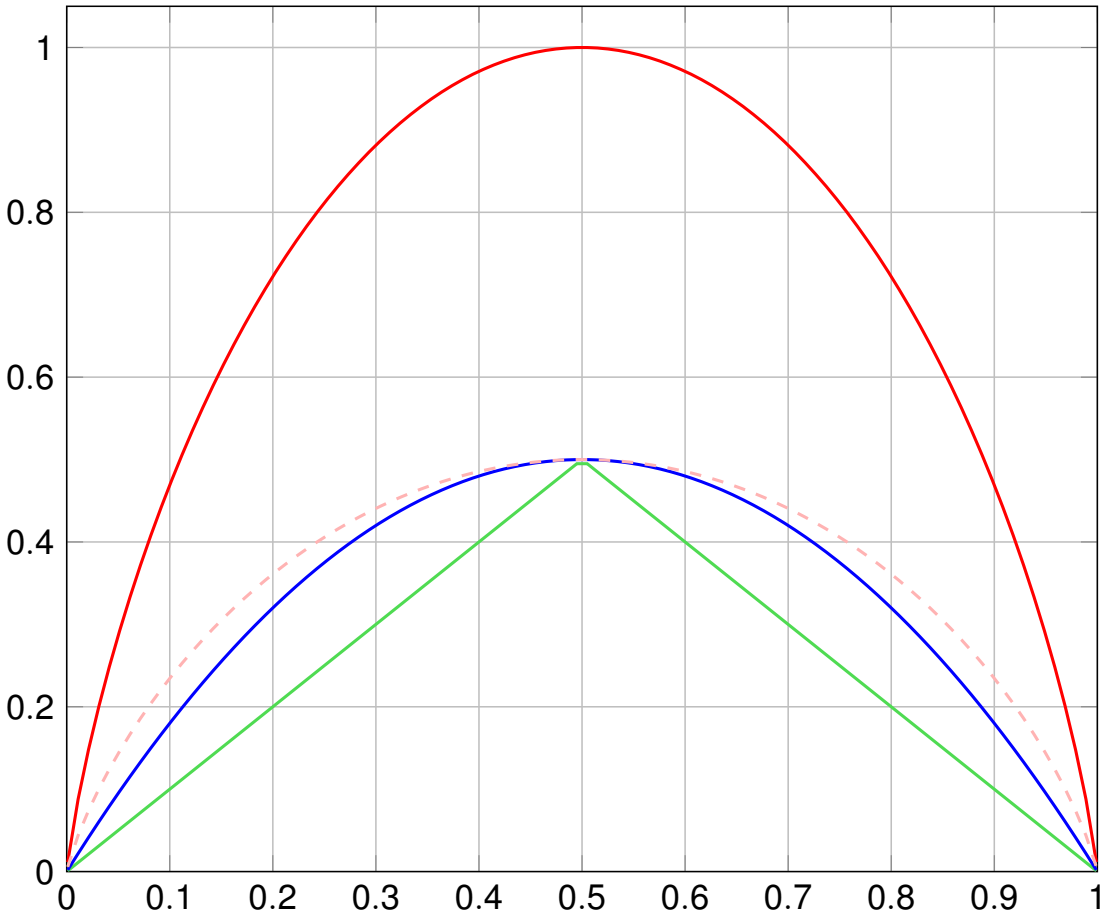


Other Gain Measures

Let $x \in [0, 1]$ be the fraction of positive training examples.

- Training Error: $\min(x, 1 - x)$
- Gini Impurity: $x(1 - x) + (1 - x)x$
- Entropy: $-x \cdot \log_2(x) - (1 - x) \cdot \log_2(1 - x)$

Other Gain Measures



Other Gain Measures

Let $x \in [0, 1]$ be the fraction of positive training examples.

- Training Error: $\min(x, 1 - x)$
- Gini Impurity: $x(1 - x) + (1 - x)x$
- Entropy: $-x \cdot \log_2(x) - (1 - x) \cdot \log_2(1 - x)$

Example: Entropy vs. Training Error

x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Example: Entropy vs. Training Error

x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Example: Entropy vs. Training Error

x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Example: Entropy vs. Training Error

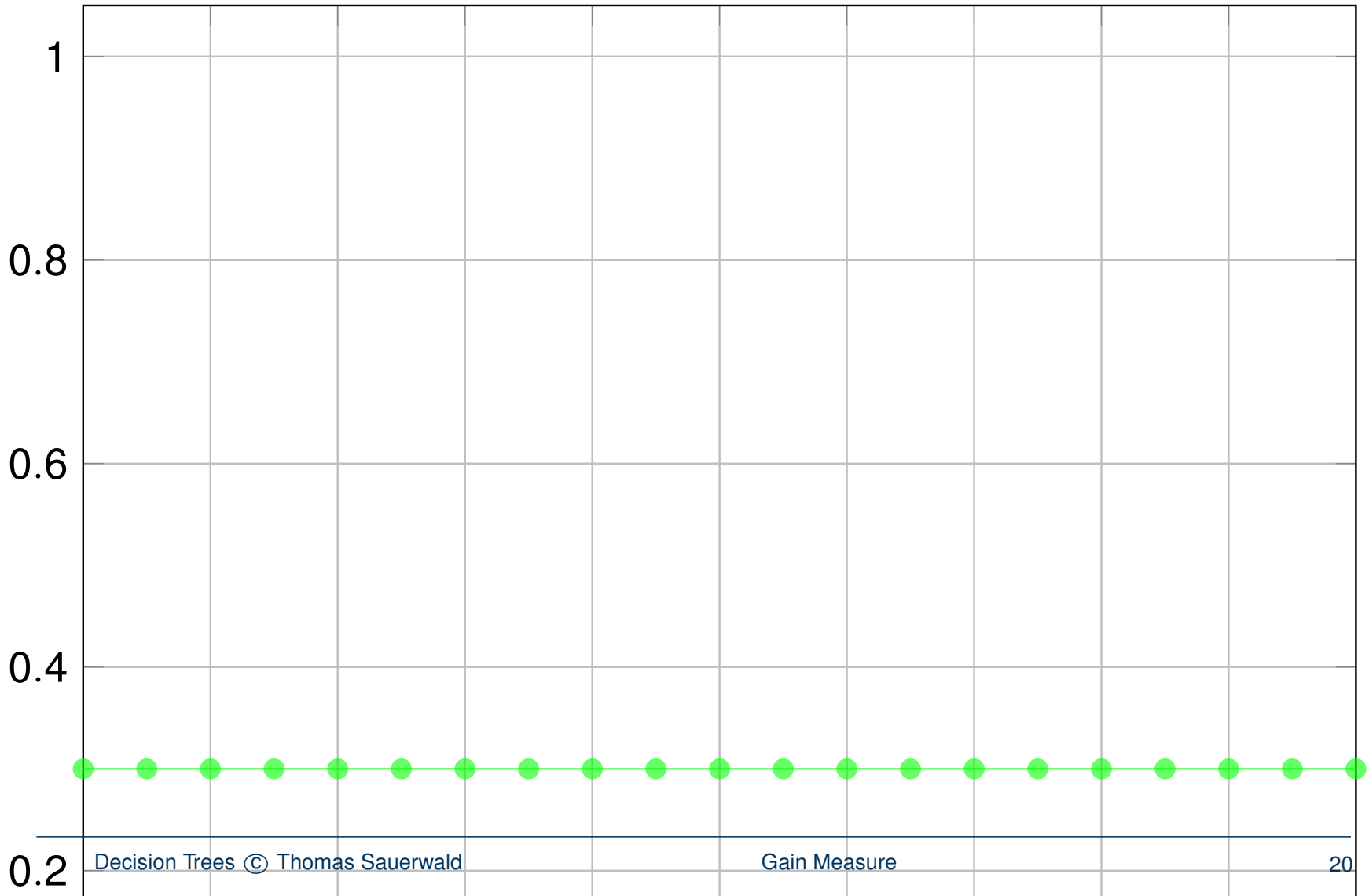
x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0

Example: Entropy vs. Training Error

x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0

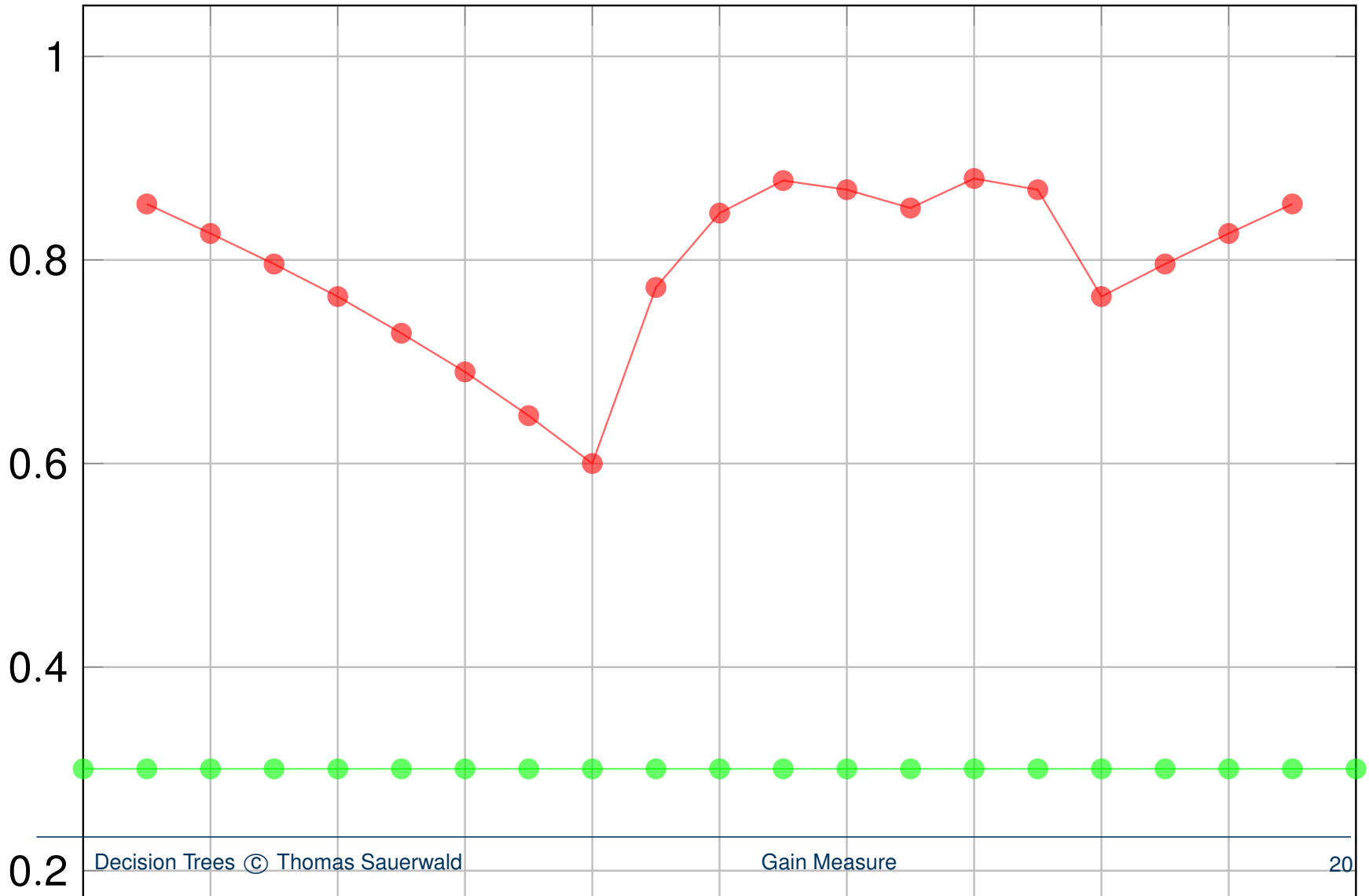
Example: Entropy vs. Training Error

x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0



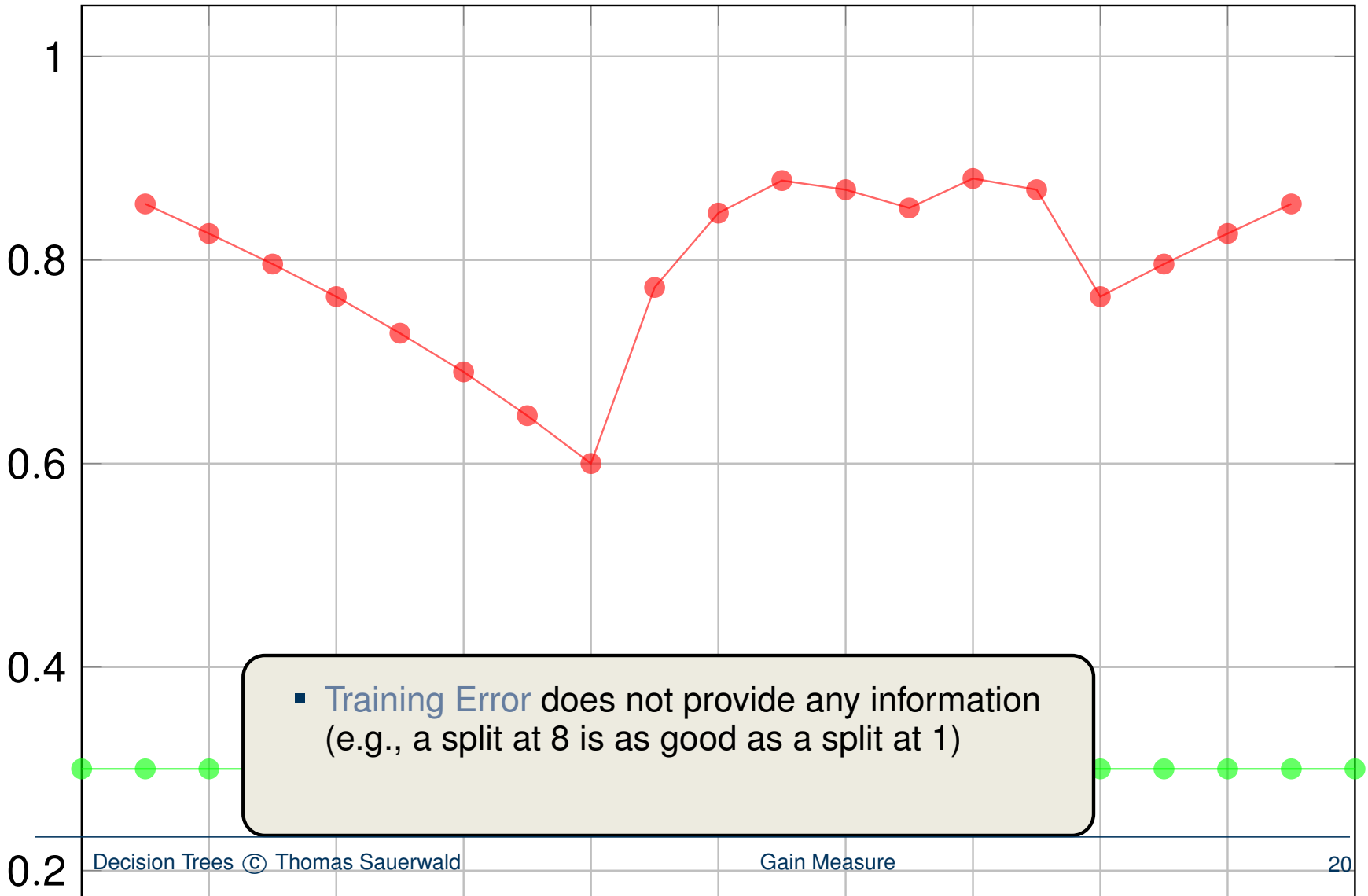
Example: Entropy vs. Training Error

x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0



Example: Entropy vs. Training Error

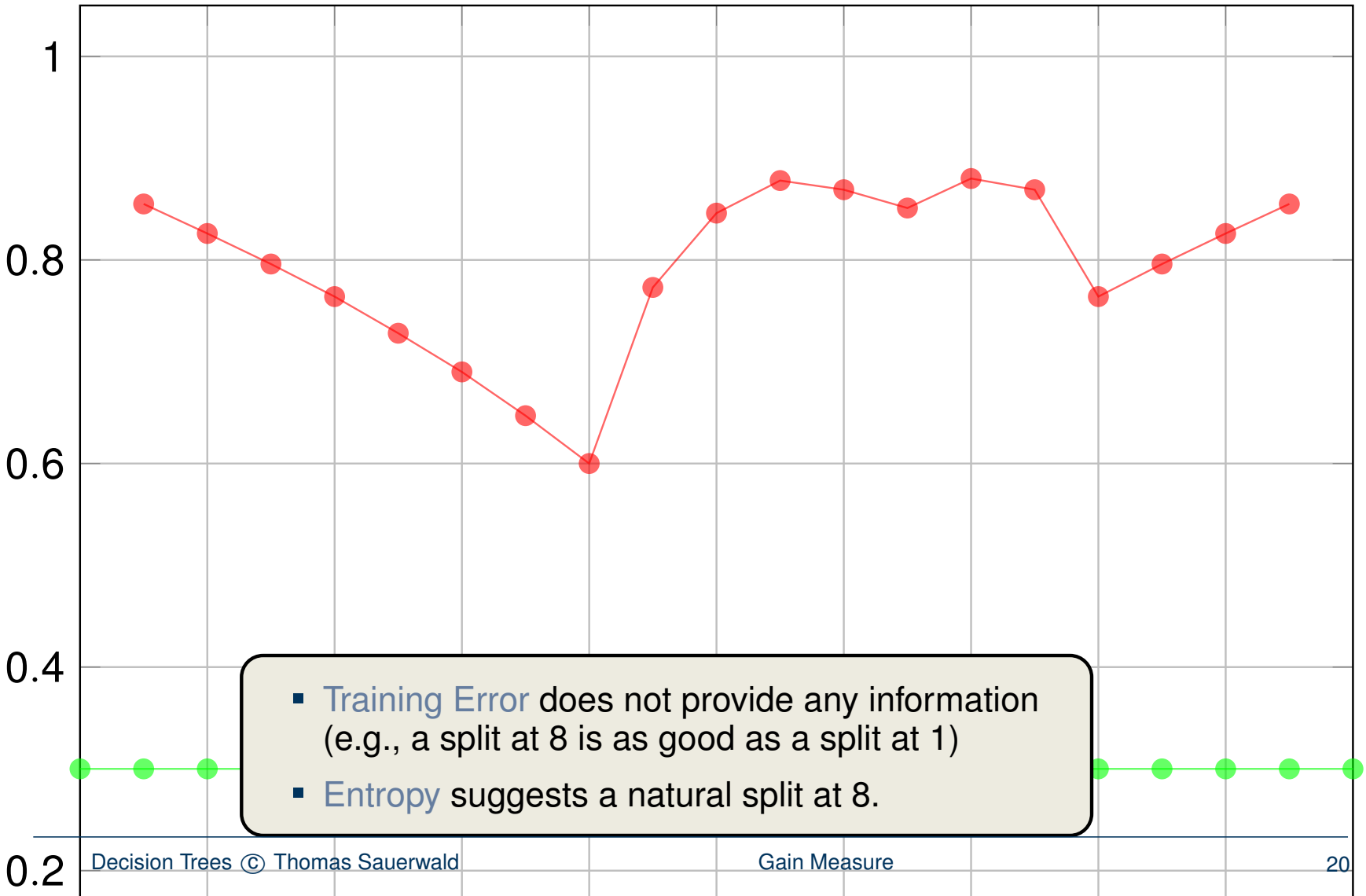
x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0



- Training Error does not provide any information (e.g., a split at 8 is as good as a split at 1)

Example: Entropy vs. Training Error

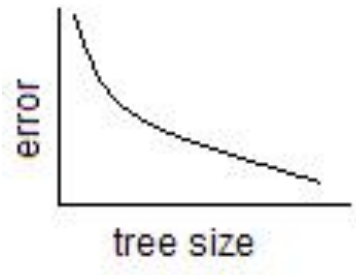
x_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y_i	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0



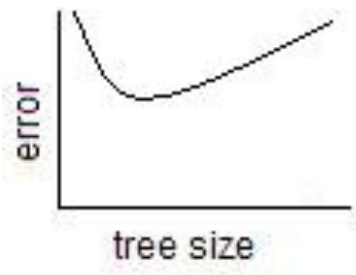
- Training Error does not provide any information (e.g., a split at 8 is as good as a split at 1)
- Entropy suggests a natural split at 8.

Outline

Training vs. Test Error and Overfitting (1/2)



tree size vs. training error

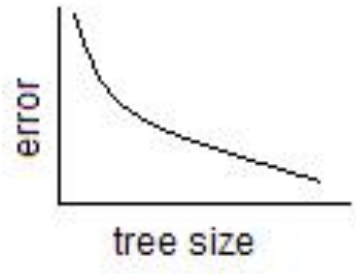


tree size vs. testing error

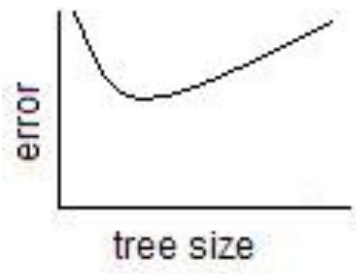
- training error decreases with tree size

source: Lecture notes by Robert Shapira

Training vs. Test Error and Overfitting (1/2)



tree size vs. training error

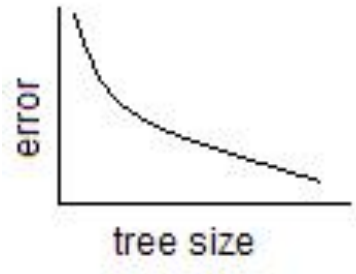


tree size vs. testing error

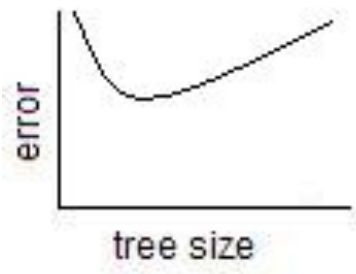
- training error decreases with tree size
- test error decreases with tree size (at first!)

source: Lecture notes by Robert Shapira

Training vs. Test Error and Overfitting (1/2)



tree size vs. training error

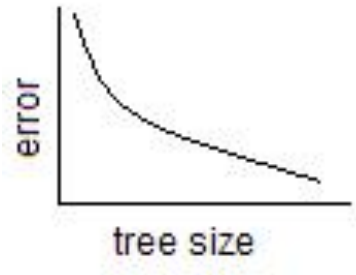


tree size vs. testing error

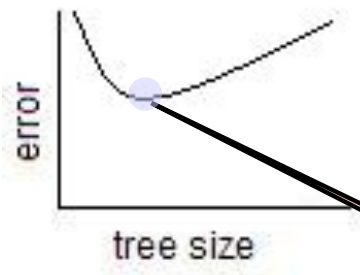
- training error decreases with tree size
- test error decreases with tree size (at first!)
- However, if tree size gets to large, decision tree does not fit the data but fits the noise in the data.
  $\Rightarrow$ **Overfitting!!!**

source: Lecture notes by Robert Shapira

Training vs. Test Error and Overfitting (1/2)



tree size vs. training error



tree size vs. testing error

source: Lecture notes by Robert Shapira

- training error decreases with tree size
- test error decreases with tree size (at first!)
- However, if tree size gets to large, decision tree does not fit the data but fits the noise in the data.
  $\Rightarrow$ **Overfitting!!!**

Rule-of-thumb: the point where we neither underfit nor overfit is when the test error is minimised.

Training vs. Test Error and Overfitting (2/2)

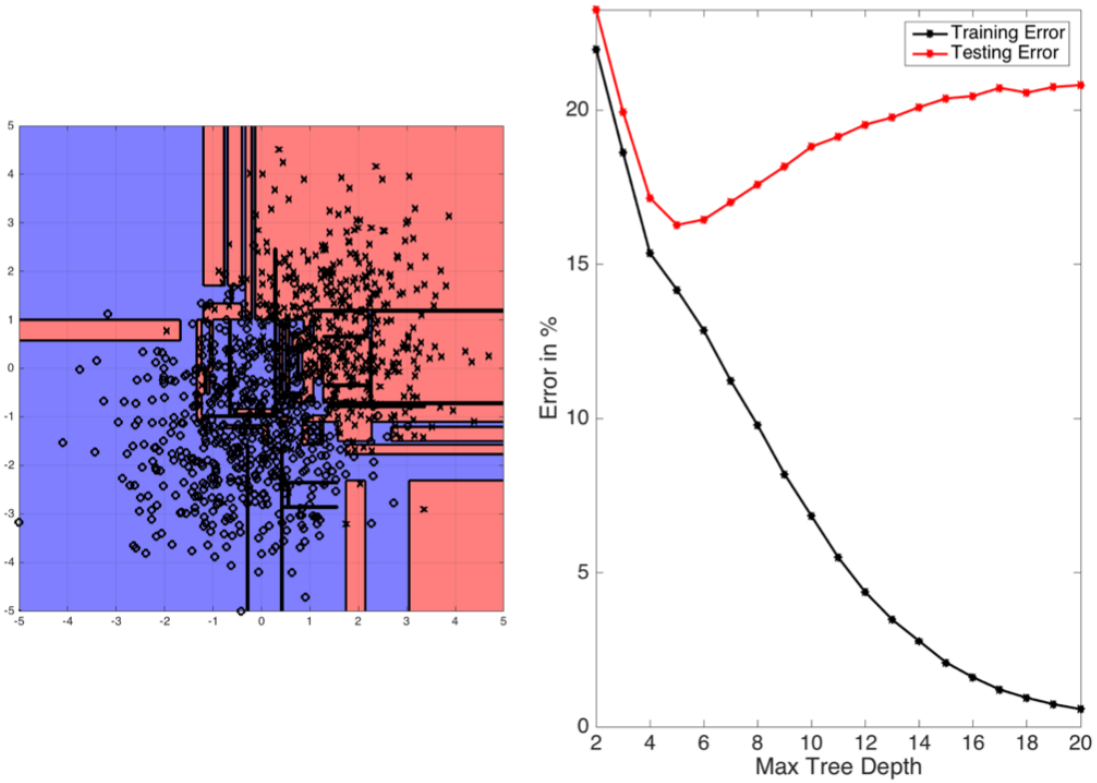


Fig: ID3-trees are prone to overfitting as the tree depth increases. The left plot shows the learned decision boundary of a binary data set drawn from two Gaussian distributions. The right plot shows the testing and training errors with increasing tree depth.

Source: Kilian Weinberger

Exercise: Suboptimality of ID3

18.2 **(Suboptimality of ID3)**
Consider the following training set, where $\mathcal{X} = \{0,1\}^3$ and $\mathcal{Y} = \{0,1\}$:

$$((1,1,1),1)$$
$$((1,0,0),1)$$
$$((1,1,0),0)$$
$$((0,0,1),0)$$

Suppose we wish to use this training set in order to build a decision tree of depth 2 (i.e., for each input we are allowed to ask two questions of the form ($x_i = 0$?) before deciding on the label).

1. Suppose we run the ID3 algorithm up to depth 2 (namely, we pick the root node and its children according to the algorithm, but instead of keeping on with the recursion, we stop and pick leaves according to the majority label in each subtree). Assume that the subroutine used to measure the quality of each feature is based on the entropy function (so we measure the *information gain*), and that if two features get the same score, one of them is picked arbitrarily. Show that the training error of the resulting decision tree is at least $1/4$.
2. Find a decision tree of depth 2 that attains zero training error.

taken from Shalev-Schwartz, Ben-David

Summary: Decision Trees

+ easy to implement and evaluate
+ simple to understand (by humans)
+ can be used for both classification and regression
+ do not require feature normalisation like k-NN
+ several tricks to increase efficiency through parallel computation:
  - When searching for best possible split, features can be investigated in parallel
  - Process of sorting and computing accumulated sums can be parallelised
  - Also nodes in different branches can be computed in parallel

– computationally hard to find “optimal” decision trees
– issues with overfitting
– cannot find diagonal (fix: multivariate decision trees) or curved boundaries
– small change in data could lead to large change in the decision tree (fix: use bagging/random forests)

Extension of Decision Trees: Random Forest

Input: data set S with m data points, d features

Random Forests

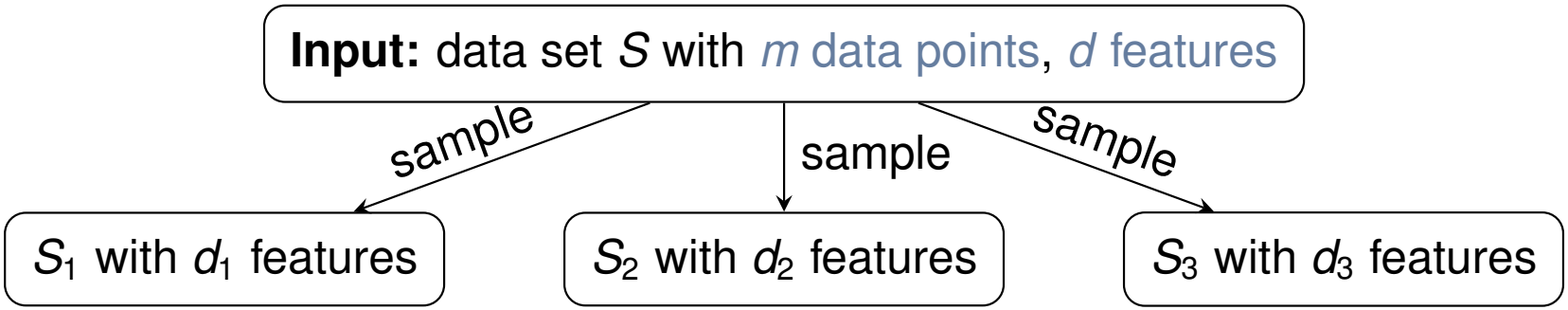
Extension of Decision Trees: Random Forest

Input: data set S with m data points, d features

Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features

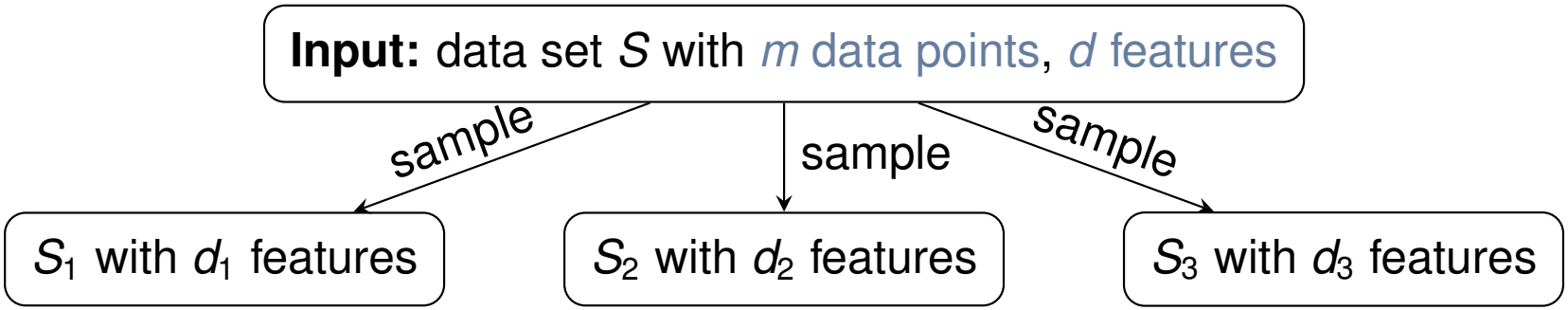
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features

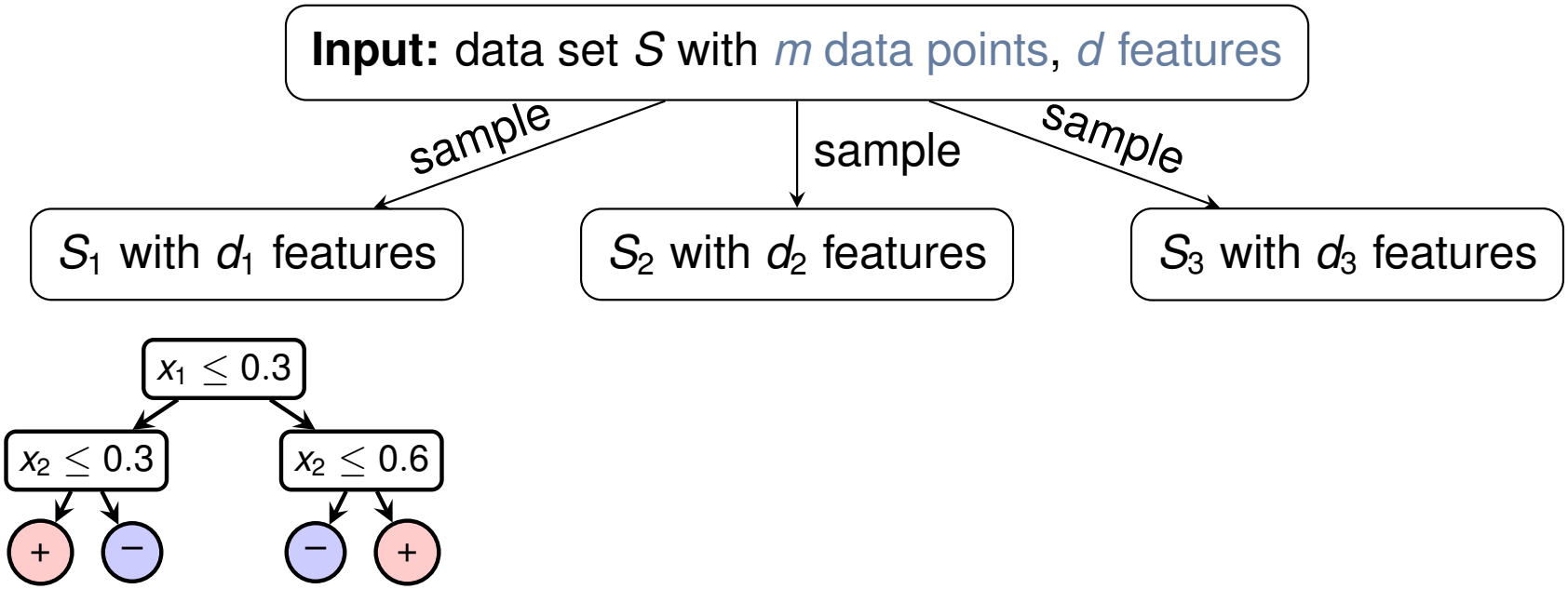
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree

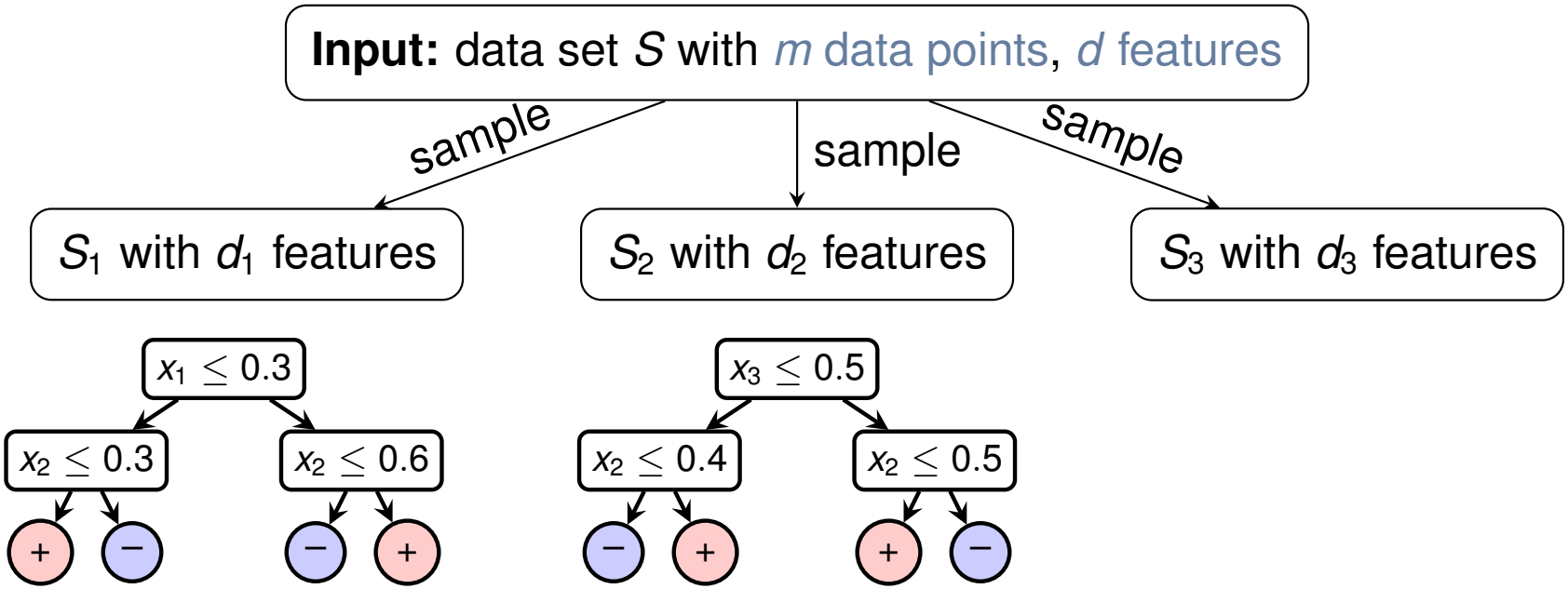
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree

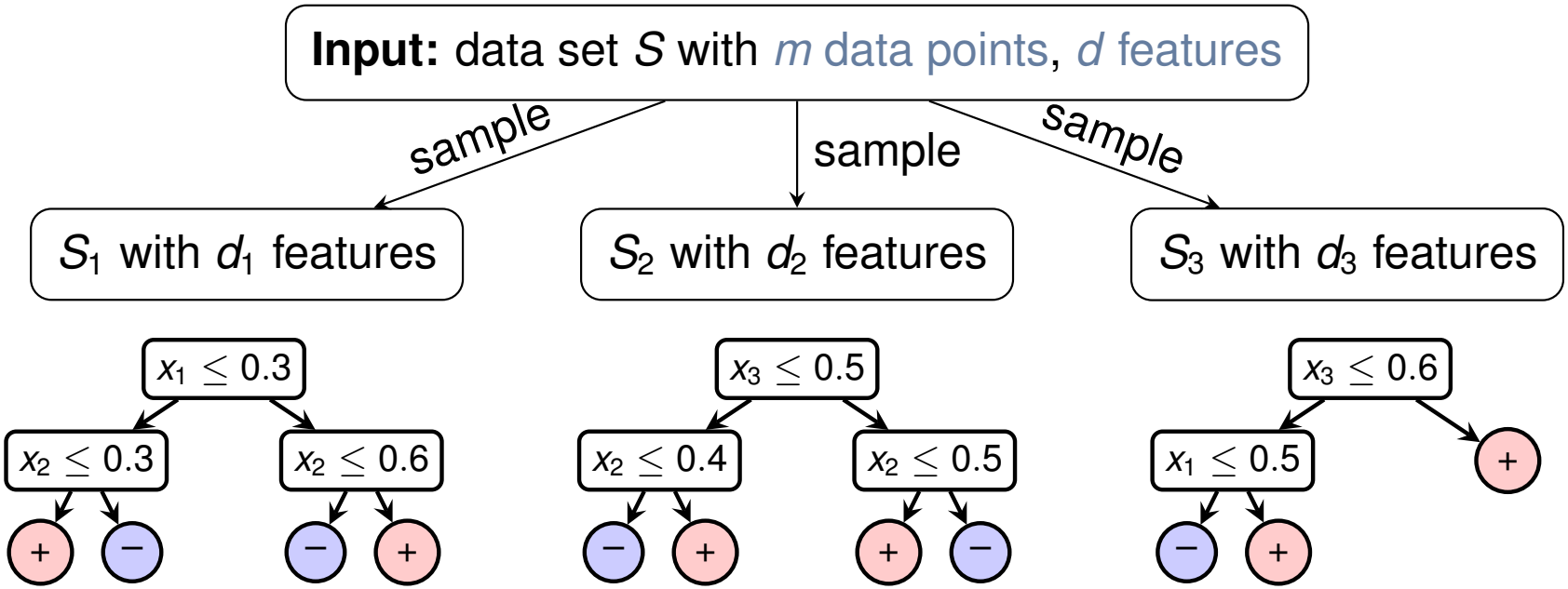
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree

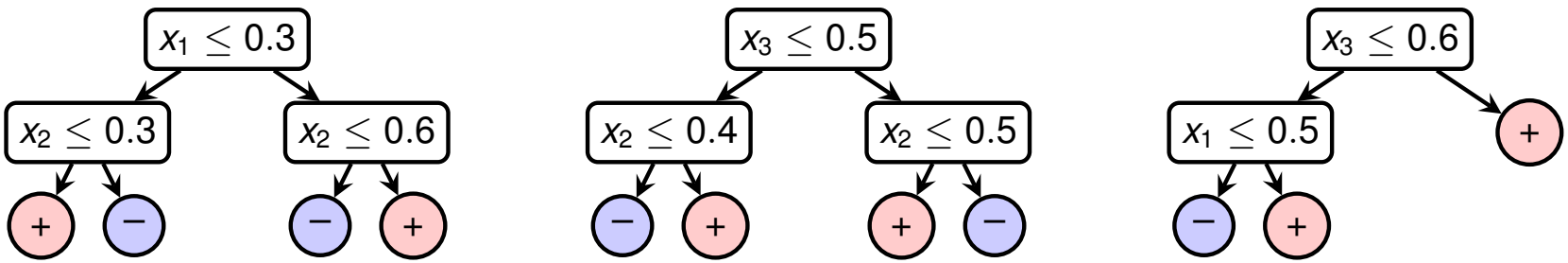
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree

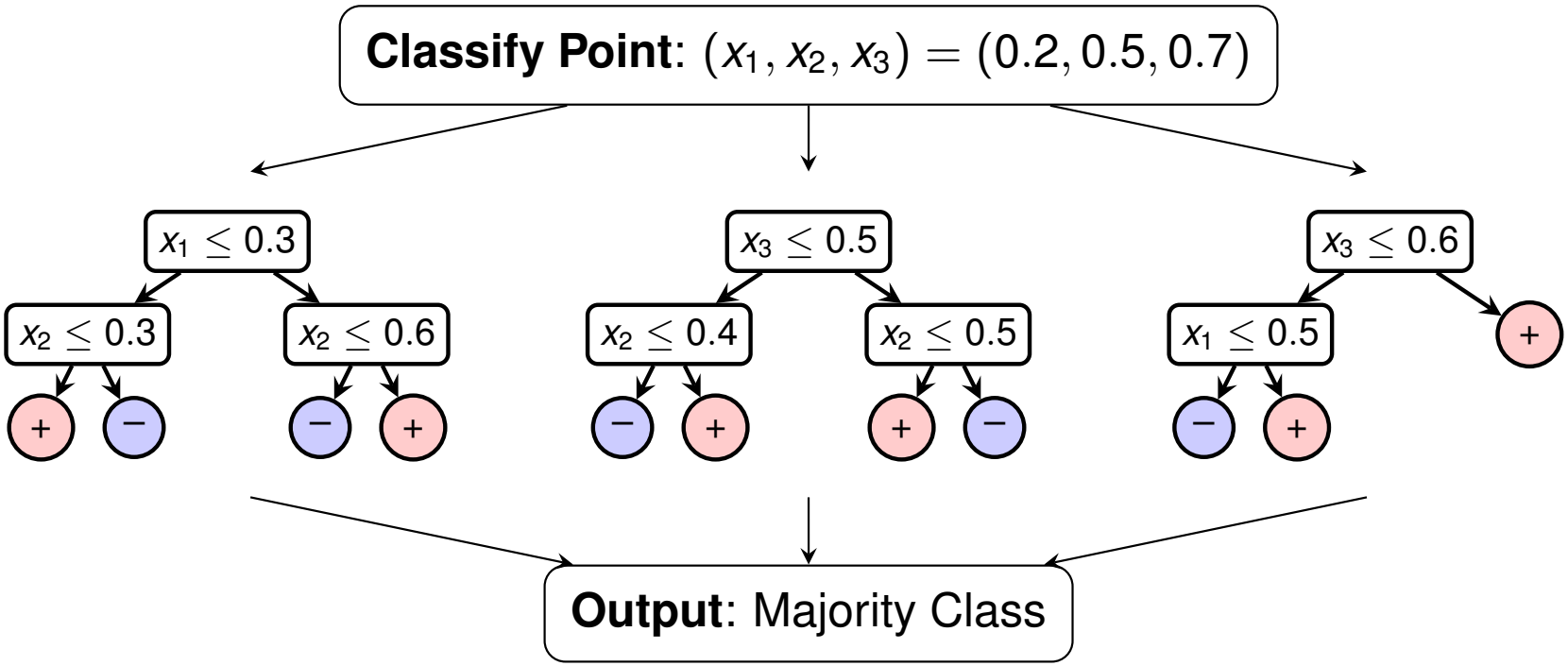
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

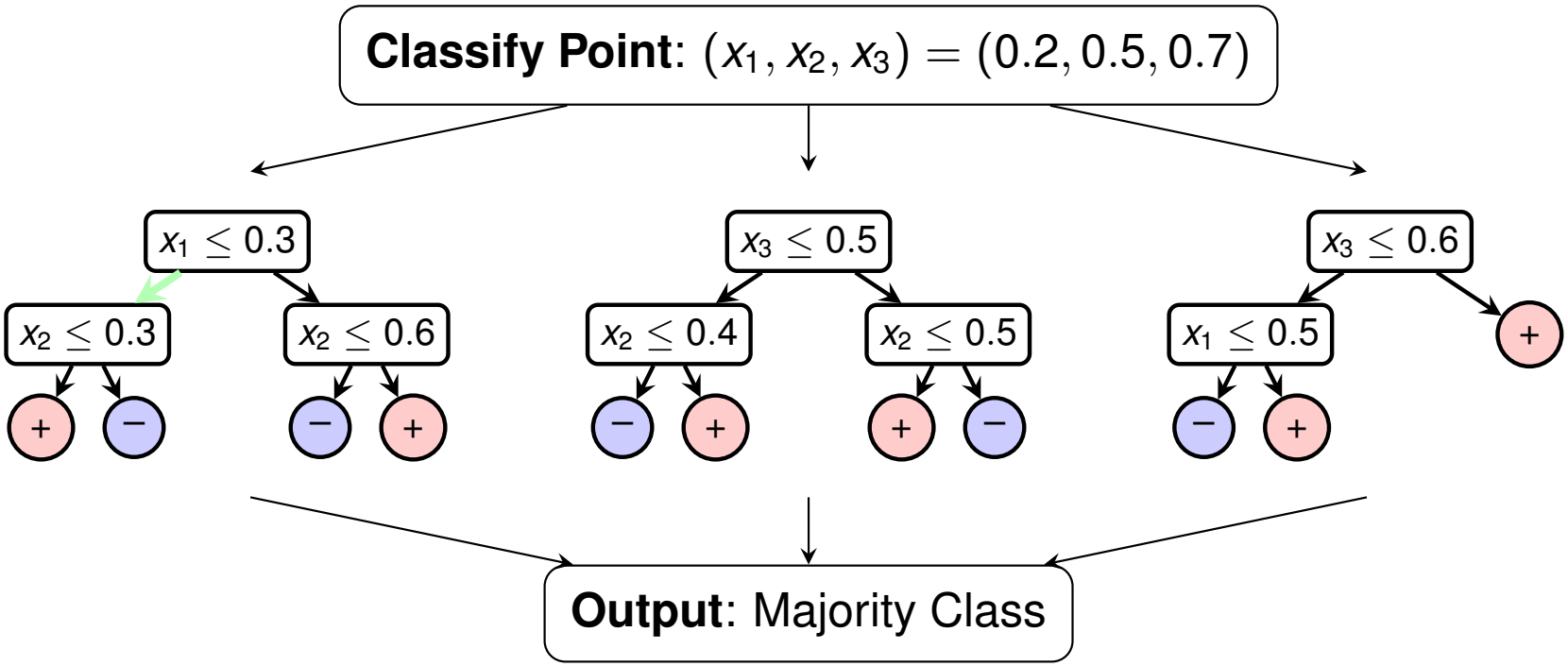
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

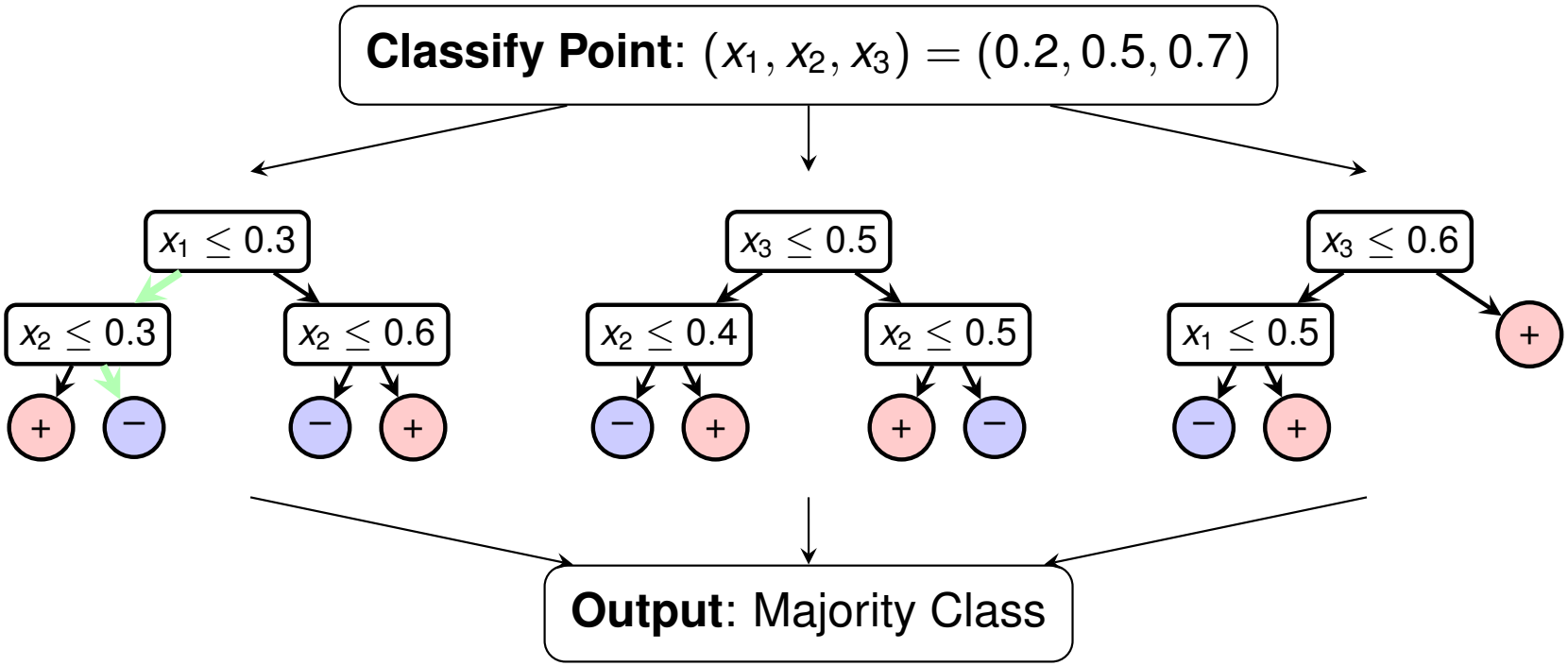
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

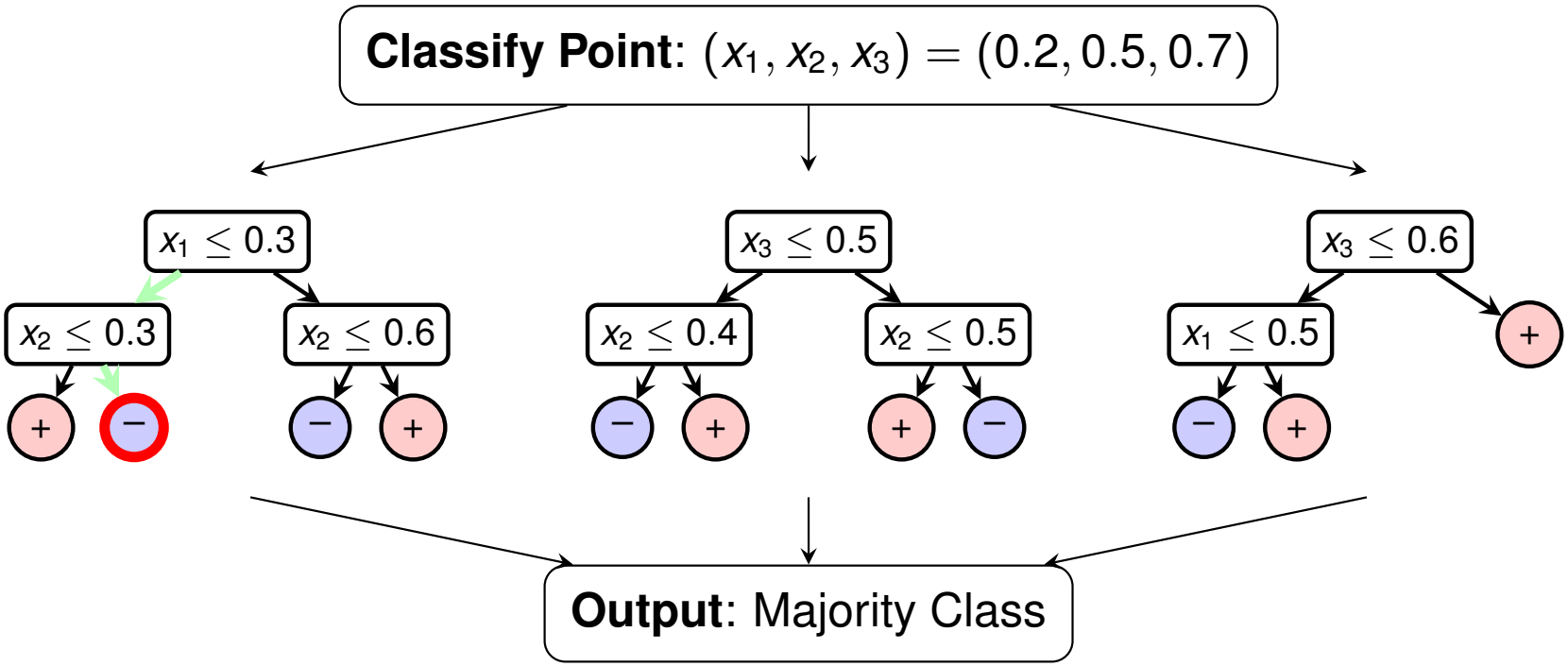
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

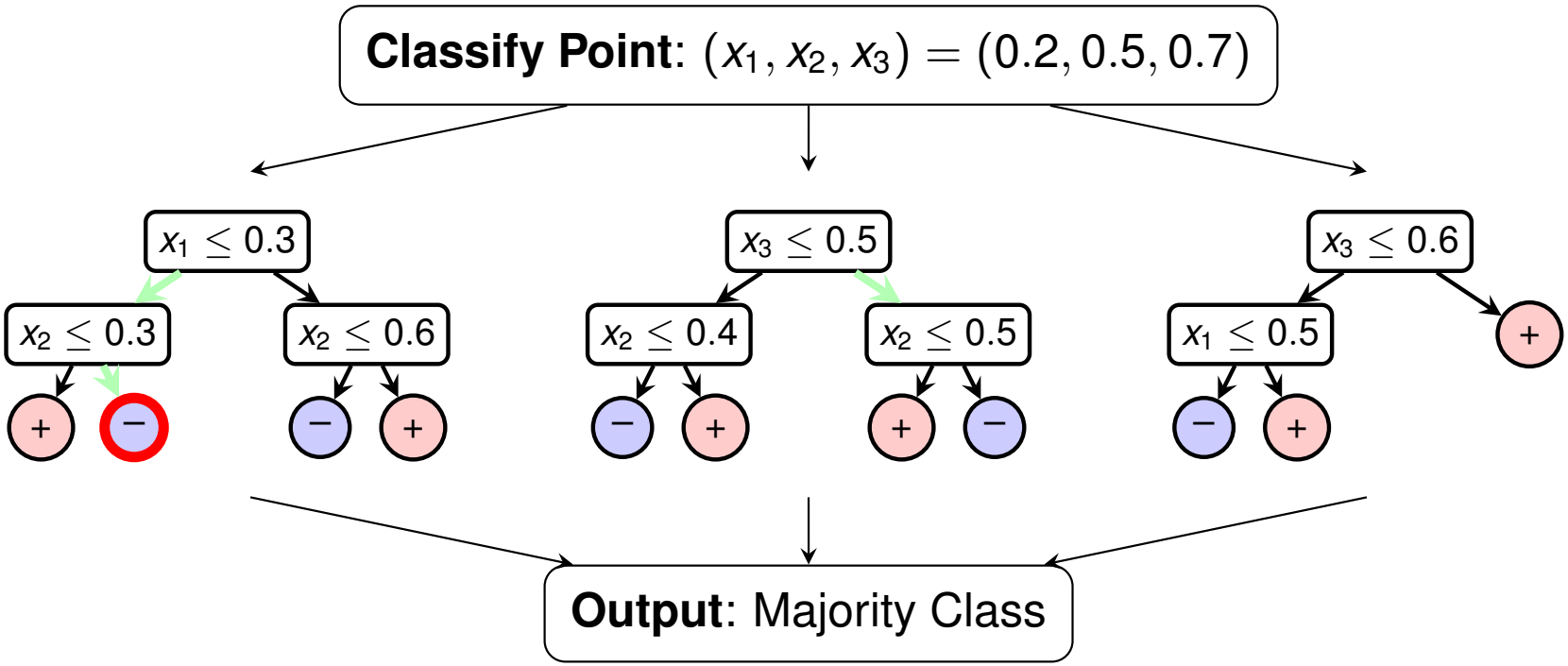
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

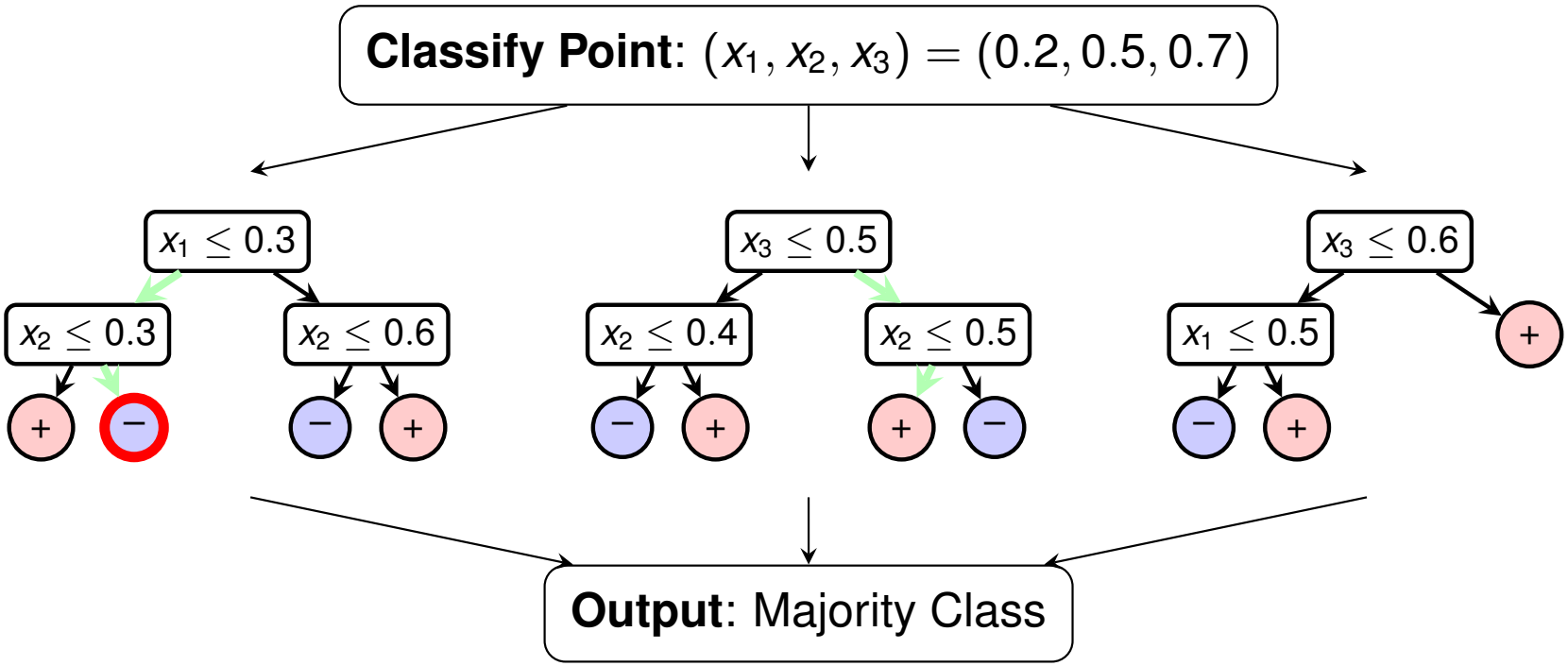
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

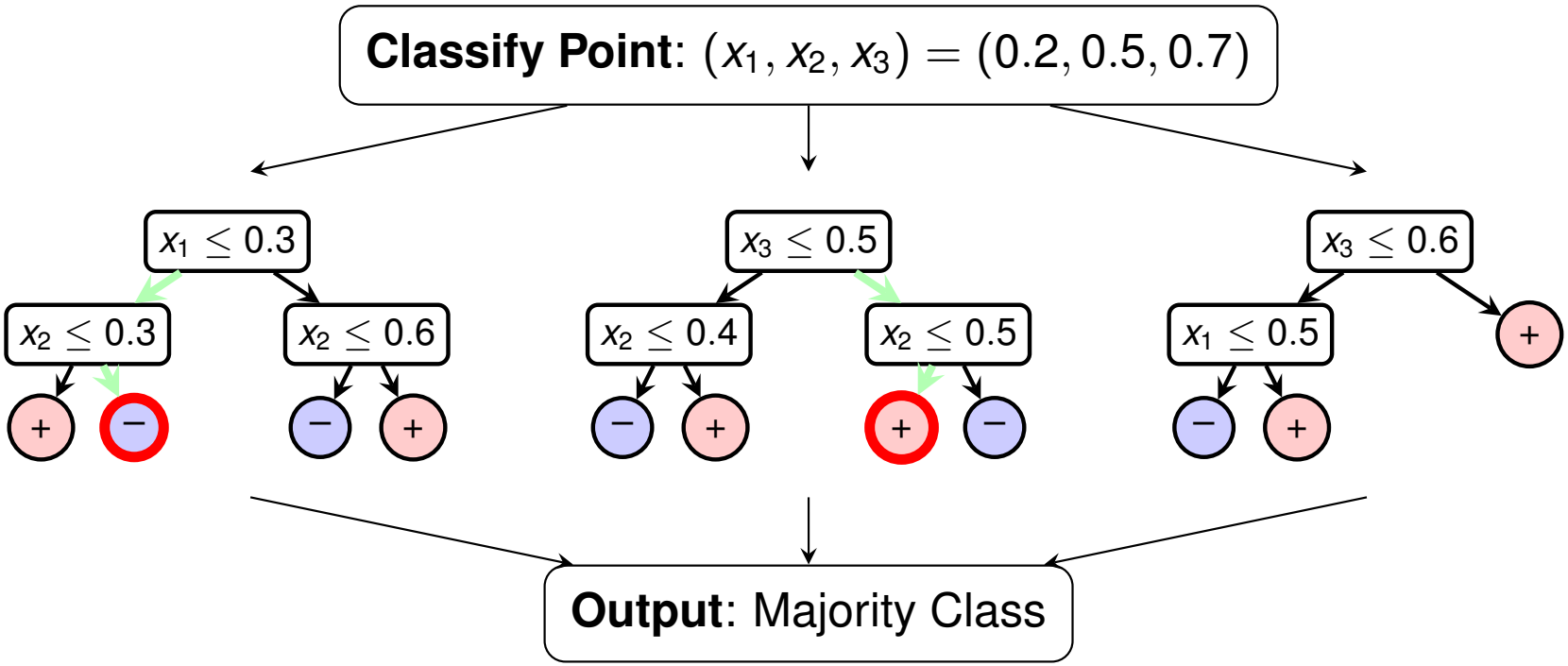
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

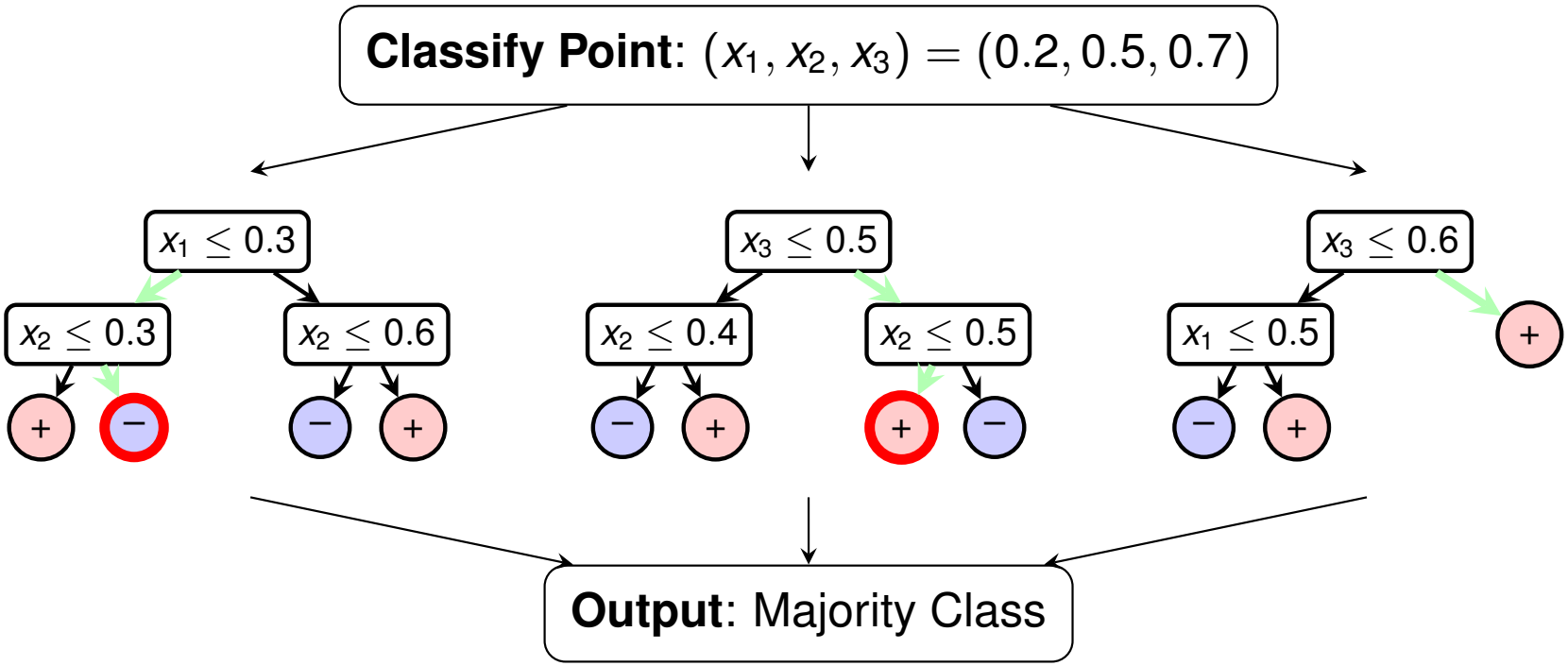
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

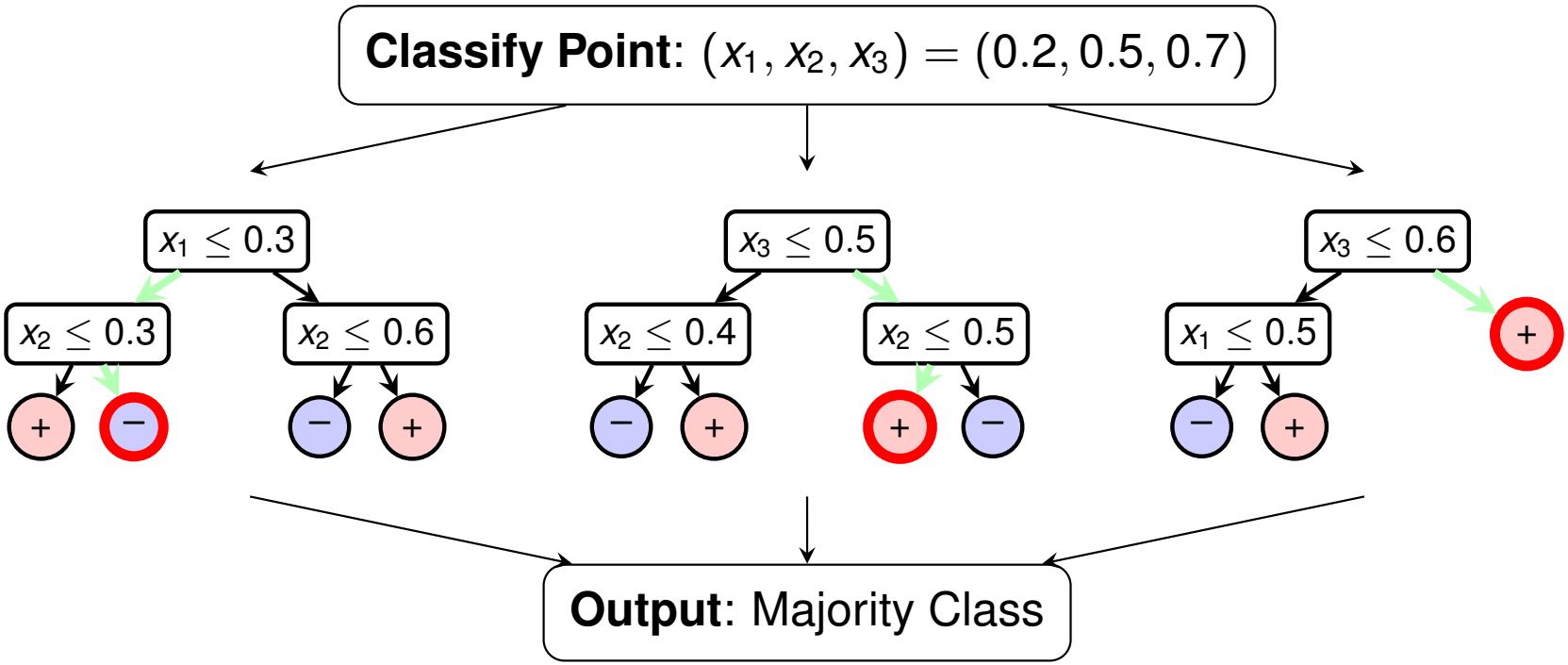
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

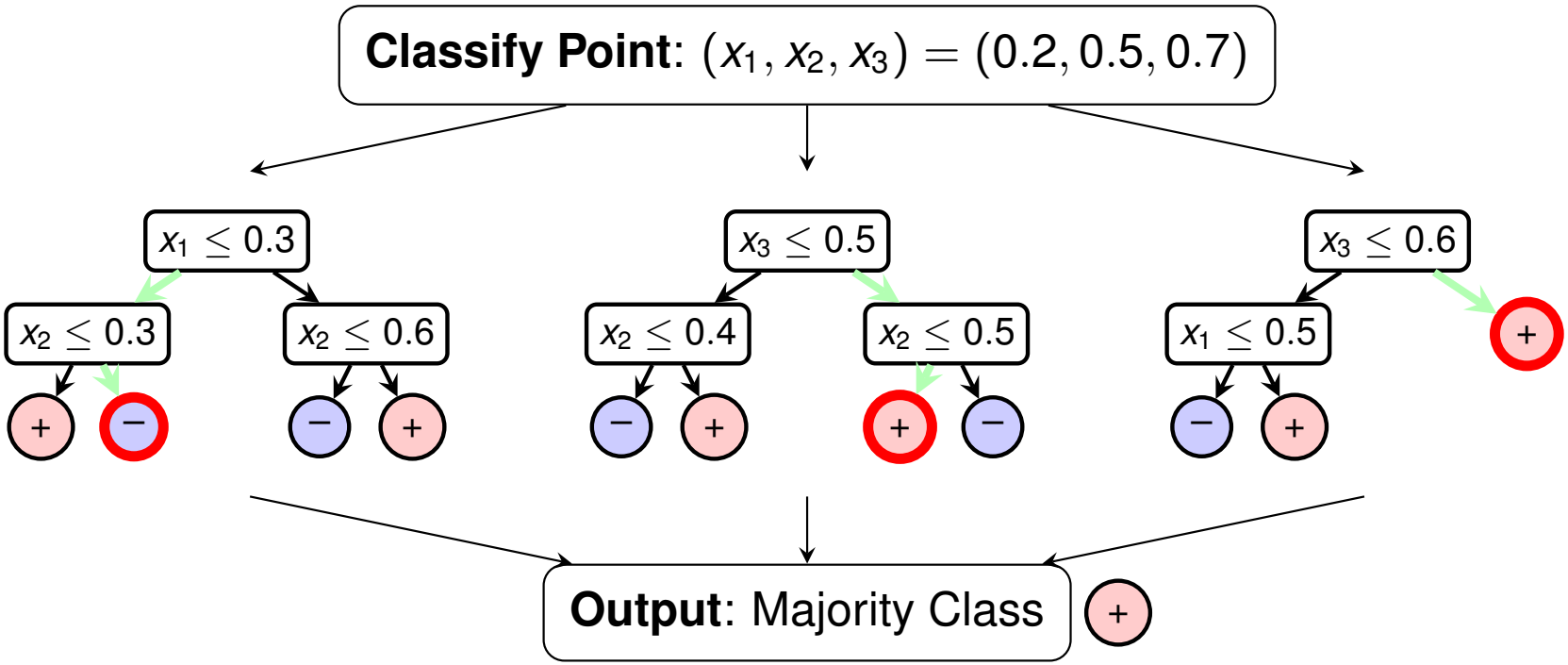
Extension of Decision Trees: Random Forest



Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

Extension of Decision Trees: Random Forest

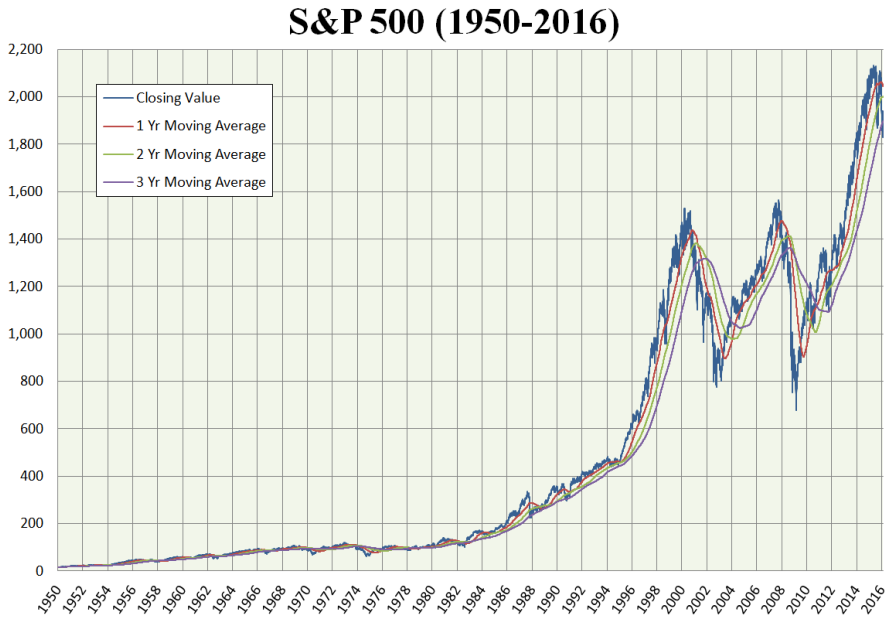


Random Forests

1. **Bootstrap**: For each tree, sample m points from S with replacement
2. **Feature Bagging**: For each tree, select a random set of d' features
3. **Build Trees**: Train each tree
4. **Aggregation**: For classification, take majority vote over trees

Outline

S & P 500



source: Wikipedia

S & P 500

- "stock market index that measures the stock performance of 500 large companies listed on stock exchanges in the US" (Wikipedia)
- However it also contains several multi-national companies
- composition is decided by a committee
- on average annual return of 9.6% since 1926
- data set with daily prices since 1928 available on finance.yahoo.com

The Data Set

```
                                        head sp500.csv
Date,Open,High,Low,Close,Adj Close,Volume
1927-12-30,17.660000,17.660000,17.660000,17.660000,17.660000,0
1928-01-03,17.760000,17.760000,17.760000,17.760000,17.760000,0
1928-01-04,17.719999,17.719999,17.719999,17.719999,17.719999,0
1928-01-05,17.549999,17.549999,17.549999,17.549999,17.549999,0
1928-01-06,17.660000,17.660000,17.660000,17.660000,17.660000,0
1928-01-09,17.500000,17.500000,17.500000,17.500000,17.500000,0
1928-01-10,17.370001,17.370001,17.370001,17.370001,17.370001,0
1928-01-11,17.350000,17.350000,17.350000,17.350000,17.350000,0
1928-01-12,17.469999,17.469999,17.469999,17.469999,17.469999,0
                                        tail sp500.csv
2020-04-01,2498.080078,2522.750000,2447.489990,2470.500000,2470.500000,5947900000
2020-04-02,2458.540039,2533.219971,2455.790039,2526.899902,2526.899902,6454990000
2020-04-03,2514.919922,2538.179932,2459.959961,2488.649902,2488.649902,6087190000
2020-04-06,2578.280029,2676.850098,2574.570068,2663.679932,2663.679932,6391860000
2020-04-07,2738.649902,2756.889893,2657.669922,2659.409912,2659.409912,7040720000
2020-04-08,2685.000000,2760.750000,2663.300049,2749.979980,2749.979980,5856370000
2020-04-09,2776.989990,2818.570068,2762.360107,2789.820068,2789.820068,7880140000
2020-04-13,2782.459961,2782.459961,2721.169922,2761.629883,2761.629883,5274310000
2020-04-14,2805.100098,2851.850098,2805.100098,2846.060059,2846.060059,5567400000
2020-04-15,2795.639893,2801.879883,2761.540039,2783.360107,2783.360107,5203390000
```

In total, more than 23,000 timesteps!

The Set-up

- use a regression-tree to decide each day whether to go long or go short:
  - if we go long and S&P changes by $1 + \Delta$, our capital changes by $1 + \Delta$
  - if we go short and S&P changes by $1 - \Delta$, our capital changes by $1 + \Delta$

The Set-up

We make binary predictions but experience numerical reward!

- use a regression-tree to decide each day whether to go long or go short:
  - if we go long and S&P changes by $1 + \Delta$, our capital changes by $1 + \Delta$
  - if we go short and S&P changes by $1 - \Delta$, our capital changes by $1 + \Delta$

The Set-up

We make binary predictions but experience numerical reward!

- use a regression-tree to decide each day whether to go long or go short:
  - if we go long and S&P changes by $1 + \Delta$, our capital changes by $1 + \Delta$
  - if we go short and S&P changes by $1 - \Delta$, our capital changes by $1 + \Delta$
- As features, we choose:
  0. Price change on previous day
  1. Weekday (Monday, Tuesday, Wednesday, Thursday or Friday)
  2. Month
  3. Distance of last price to highest price (all-time high) in percent
  4. Distance of last price to 90 day moving average in percent
  5. Distance of last price to 200 day moving average in percent
  6. Last daily change of the 200 day moving average in percent

The Set-up

We make binary predictions but experience numerical reward!

- use a regression-tree to decide each day whether to go long or go short:
  - if we go long and S&P changes by $1 + \Delta$, our capital changes by $1 + \Delta$
  - if we go short and S&P changes by $1 - \Delta$, our capital changes by $1 + \Delta$
- As features, we choose:
  0. Price change on previous day
  1. Weekday (Monday, Tuesday, Wednesday, Thursday or Friday)
  2. Month
  3. Distance of last price to highest price (all-time high) in percent
  4. Distance of last price to 90 day moving average in percent
  5. Distance of last price to 200 day moving average in percent
  6. Last daily change of the 200 day moving average in percent
- Experiments show a huge dependence on the chosen features, so important to choose good ones! (otherwise: overfitting!)

Experimental Setup

- We use the 7 features described earlier
- Depth of Regression Tree is fixed at 3 or 6 (no pruning)
- Prediction based on Regression Tree of the last 3000 days

Experimental Setup

recomputed every 5 days!

- We use the 7 features described earlier
- Depth of Regression Tree is fixed at 3 or 6 (no pruning)
- Prediction based on Regression Tree of the last 3000 days

Experimental Setup

recomputed every 5 days!

- We use the 7 features described earlier
- Depth of Regression Tree is fixed at 3 or 6 (no pruning)
- Prediction based on Regression Tree of the last 3000 days

Can we perform better than the simple Buy & Hold strategy (we buy one unit of the S&P 500 in 1940 and never sell it)?

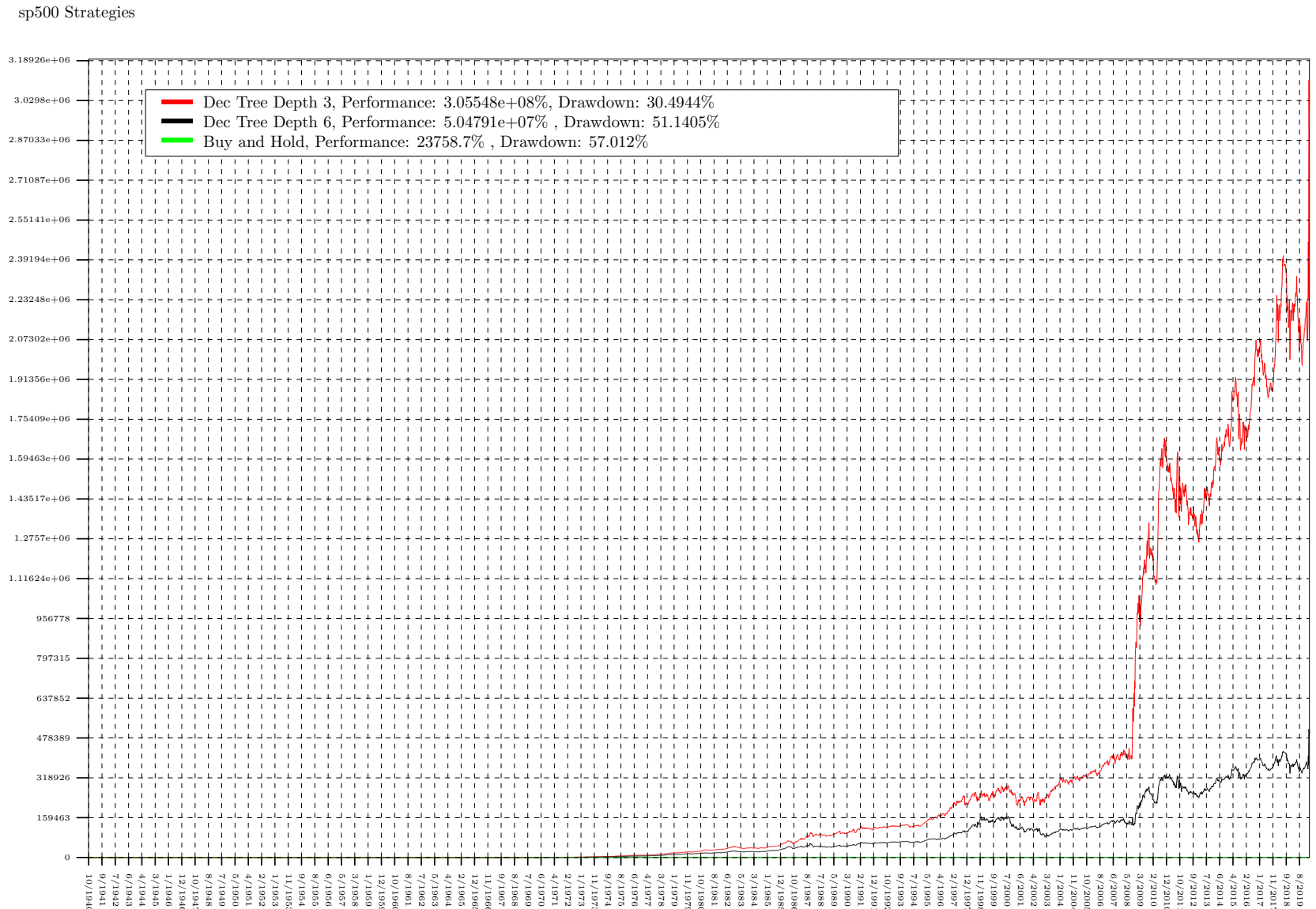
A Computed Decision Tree of Depth 3

```
*** Building DTree3 [ 14006 , 17006 ] , features: 0 to 6, maxdepth: 3 ***
Depth: 0 Best Split in [ 14006 , 17006 ) using Feature: 0 Threshold: -0.0154036 Gain: 2.03519 Type: 1
Computing Left Branch (d: 0):
Depth: 1 Best Split in [ 14006 , 17006 ) using Feature: 4 Threshold: -0.226693 Gain: 0.656874 Type: 0
Computing Left Branch (d: 1):
Depth: 2 Best Split in [ 14006 , 17006 ) using Feature: 4 Threshold: -0.278936 Gain: 0.144445 Type: 1
Computing Left Branch (d: 2):
Depth: 3 count: 1 average label: -0.0057445 prediction: -1
Computing Right Branch (d: 2):
Depth: 3 count: 9 average label: 0.138701 prediction: 1
Computing Right Branch (d: 1):
Depth: 2 Best Split in [ 14006 , 17006 ) using Feature: 6 Threshold: -0.000785874 Gain: 0.599461 Type: 0
Computing Left Branch (d: 2):
Depth: 3 count: 6 average label: 0.0377719 prediction: 1
Computing Right Branch (d: 2):
Depth: 3 count: 82 average label: -0.561689 prediction: -1
Computing Right Branch (d: 0):
Depth: 1 Best Split in [ 14006 , 17006 ) using Feature: 6 Threshold: 0.00109421 Gain: 1.93453 Type: 0
Computing Left Branch (d: 1):
Depth: 2 Best Split in [ 14006 , 17006 ) using Feature: 4 Threshold: 0.204965 Gain: 1.9395 Type: 0
Computing Left Branch (d: 2):
Depth: 3 count: 2598 average label: 1.86444 prediction: 1
Computing Right Branch (d: 2):
Depth: 3 count: 32 average label: -0.0750621 prediction: -1
Computing Right Branch (d: 1):
Depth: 2 Best Split in [ 14006 , 17006 ) using Feature: 0 Threshold: 0.00477979 Gain: 0.308498 Type: 1
Computing Left Branch (d: 2):
Depth: 3 count: 182 average label: -0.226823 prediction: -1
Computing Right Branch (d: 2):
Depth: 3 count: 90 average label: 0.0816754 prediction: 1
*** Evaluating DTree3 for Interval: ( 14006 , 17006 ) as Training Set, Interval: ( 17006 , 17011 ) as Test S
** Training Set ** Reward 2.99191 Buy-Hold: 1.25327 Clairvoyant: 18.4663
** Test Set ** Reward 0.00346266 Buy-Hold: -0.00346266 Clairvoyant: 0.0141558
```

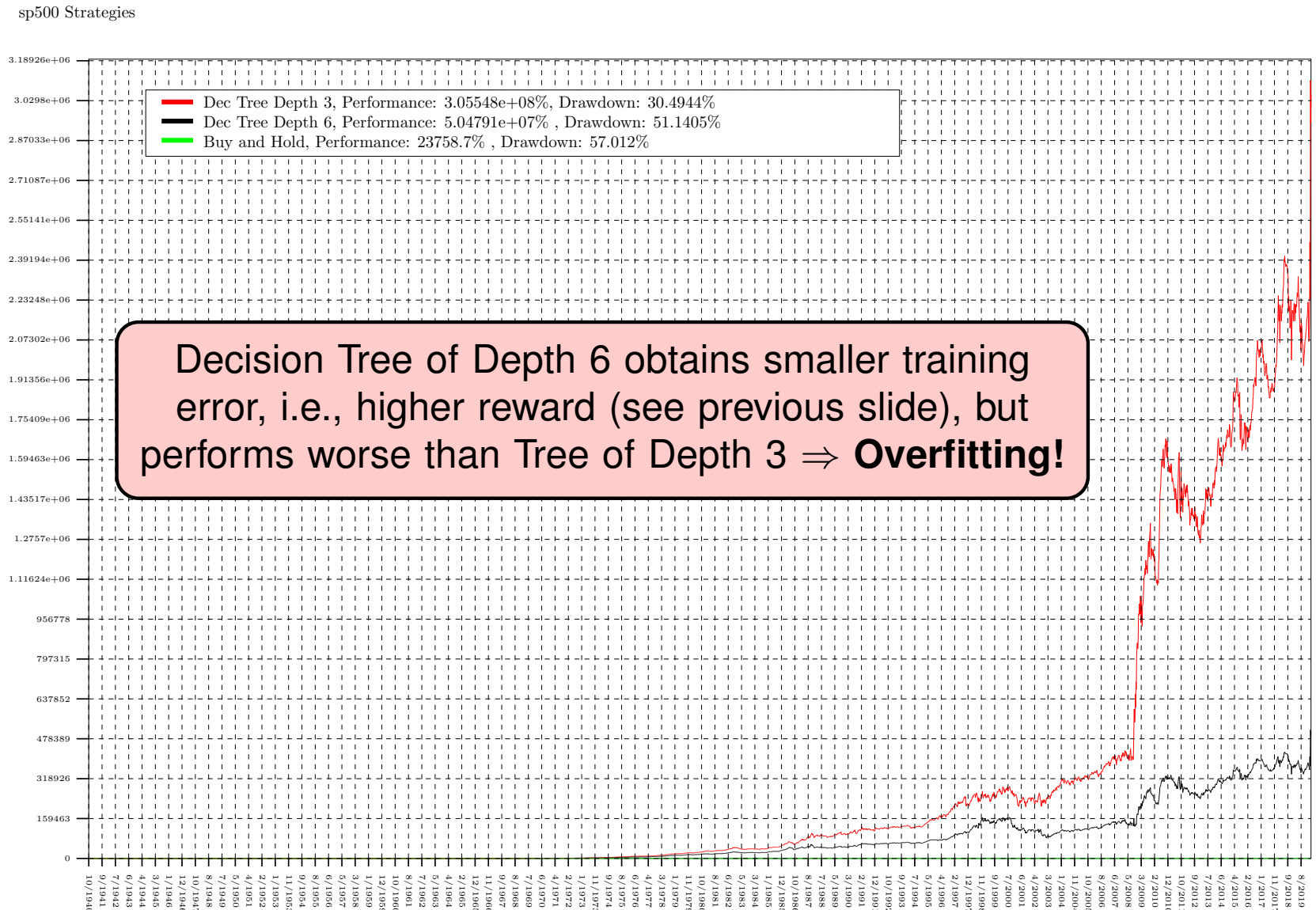
The Corresponding Decision Tree of Depth 6

```
Computing Left Branch (d: 5):
Depth: 6 count: 167 average label: -0.293149 prediction: -1
Computing Right Branch (d: 5):
Depth: 6 count: 1 average label: 0.0018293 prediction: 1
Computing Right Branch (d: 4):
Depth: 5 count: 2 average label: 0.00713538 prediction: 1
Computing Right Branch (d: 2):
Depth: 3 Best Split in [ 14006 , 17006 ) using Feature: 5 Threshold: -0.0948541 Gain: 0.19639 Type: 0
Computing Left Branch (d: 3):
Depth: 4 Best Split in [ 14006 , 17006 ) using Feature: 6 Threshold: 0.00159114 Gain: 0.161737 Type: 0
Computing Left Branch (d: 4):
Depth: 5 Best Split in [ 14006 , 17006 ) using Feature: 5 Threshold: -0.125623 Gain: 0.169434 Type: 1
Computing Left Branch (d: 5):
Depth: 6 count: 1 average label: -0.00952469 prediction: -1
Computing Right Branch (d: 5):
Depth: 6 count: 49 average label: 0.159909 prediction: 1
Computing Right Branch (d: 4):
Depth: 5 count: 3 average label: -0.0113521 prediction: -1
Computing Right Branch (d: 3):
Depth: 4 Best Split in [ 14006 , 17006 ) using Feature: 5 Threshold: -0.0737438 Gain: 0.131313 Type: 1
Computing Left Branch (d: 4):
Depth: 5 Best Split in [ 14006 , 17006 ) using Feature: 6 Threshold: 0.00147608 Gain: 0.106387 Type: 1
Computing Left Branch (d: 5):
Depth: 6 count: 25 average label: -0.100361 prediction: -1
Computing Right Branch (d: 5):
Depth: 6 count: 1 average label: 0.00602589 prediction: 1
Computing Right Branch (d: 4):
Depth: 5 Best Split in [ 14006 , 17006 ) using Feature: 0 Threshold: 0.011052 Gain: 0.0391189 Type: 0
Computing Left Branch (d: 5):
Depth: 6 count: 6 average label: 0.0380483 prediction: 1
Computing Right Branch (d: 5):
Depth: 6 count: 5 average label: -0.00107057 prediction: -1
*** Evaluating DTree6 for Interval: ( 14006 , 17006 ) as Training Set, Interval: ( 17006 , 17011 ) as Test S
** Training Set ** Reward 3.75938 Buy-Hold: 1.25327 Clairvoyant: 18.4663
** Test Set ** Reward 0.00346266 Buy-Hold: -0.00346266 Clairvoyant: 0.0141558
```

Results (Linear Scale)

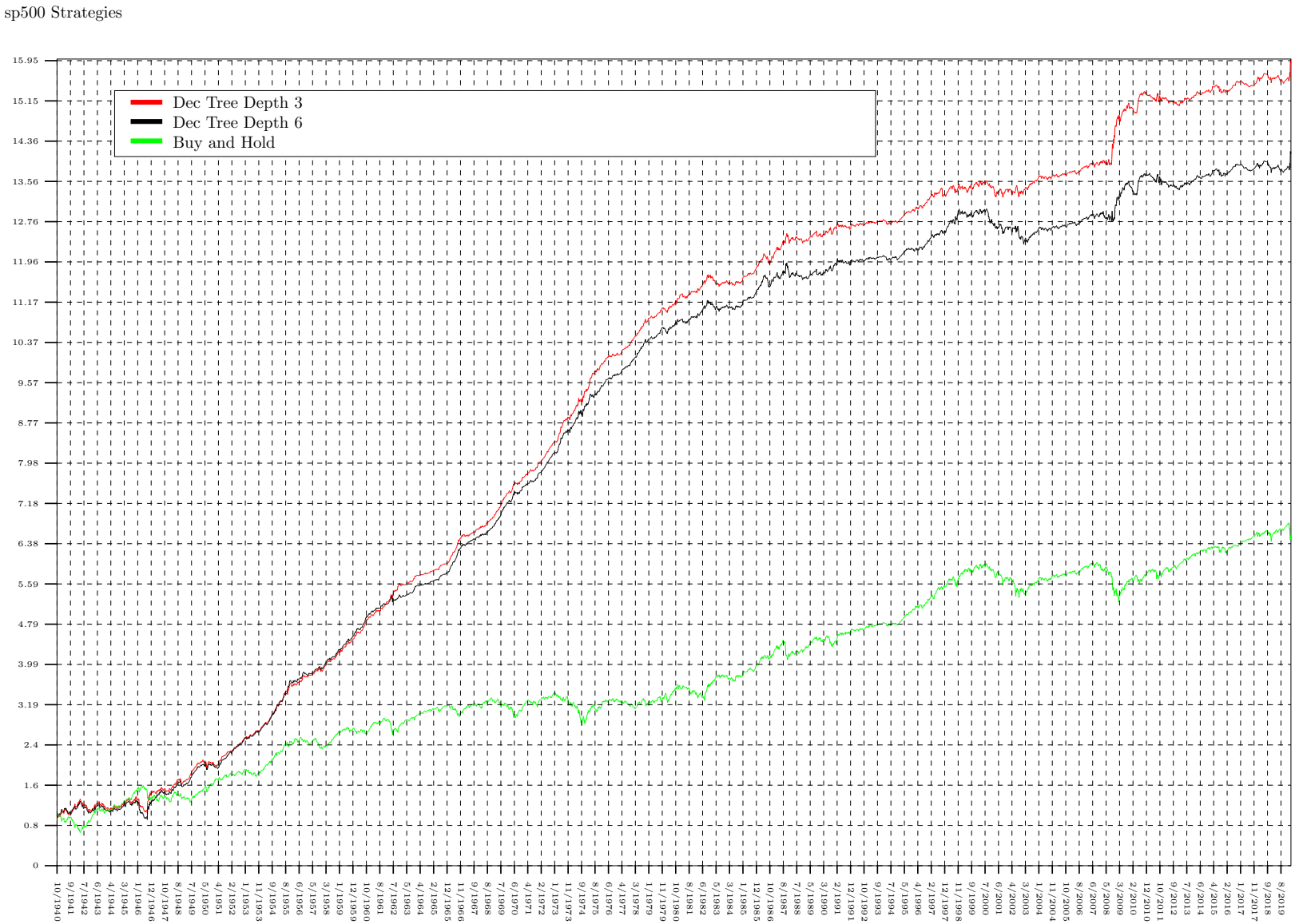


Results (Linear Scale)



Decision Tree of Depth 6 obtains smaller training error, i.e., higher reward (see previous slide), but performs worse than Tree of Depth 3 $\Rightarrow$ **Overfitting!**

Results (Logarithmic Scale)



Remarks

- Results demonstrate predictive power of regression trees
- huge number of parameters to choose from (number of features, depth, number of recent time steps etc.) (risk of overfitting!)
- More suitable tools might be random forests, SVM's or Neural Networks
- would expect even better result if instead of long and short, a third option is added (e.g., neutral)
- more economical data should be added (e.g., interest rate, unemployment rate, BIP growth, oil price, currency exchange rates etc.)
- whole simulation environment is highly simplified (e.g., we neglected transaction costs and assumed ability to compute large regression trees in 1940's(!) etc.)

Outline

The Data Set

Breast Cancer Wisconsin Dataset

- two classes (binary classification problem)
- 30 features (dimensions)
- 569 samples, 357 positive (malignant), 212 negative (benign)

The Data Set

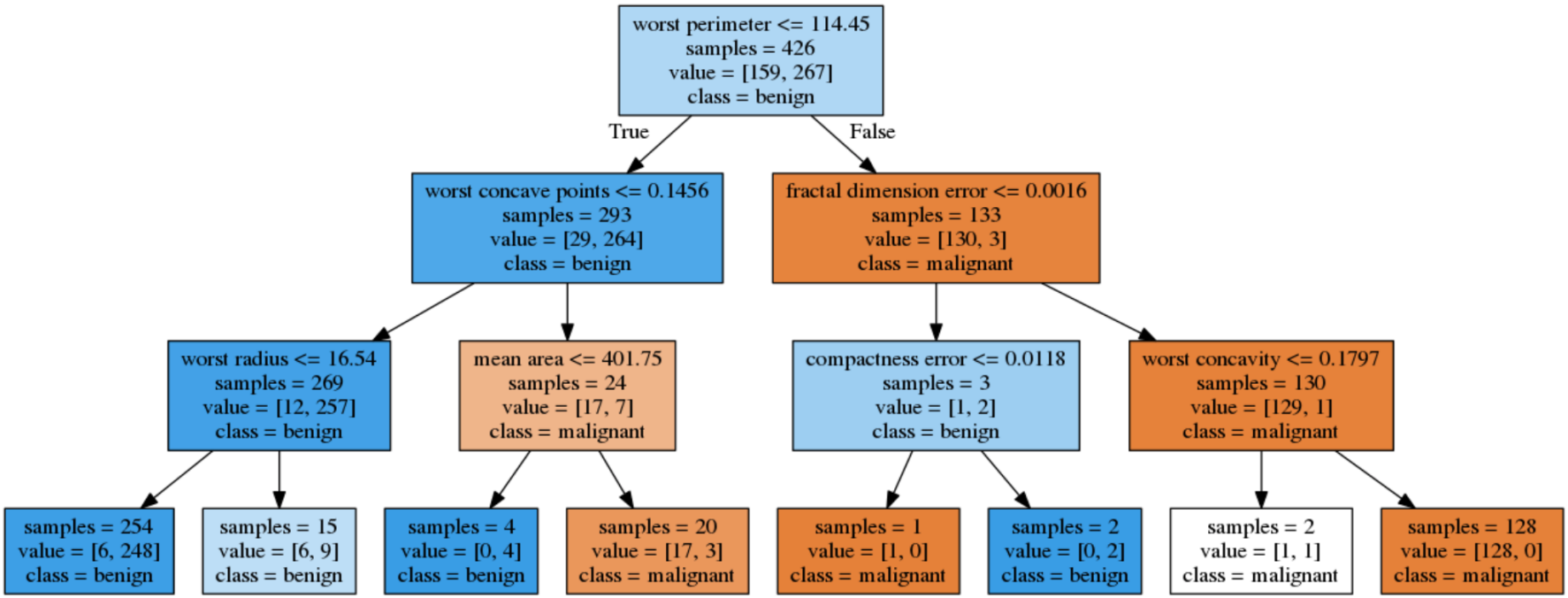
Breast Cancer Wisconsin Dataset

- two classes (binary classification problem)
- 30 features (dimensions)
- 569 samples, 357 positive (malignant), 212 negative (benign)

```
for depth in range(1, 5):
    build_tree(depth)
```

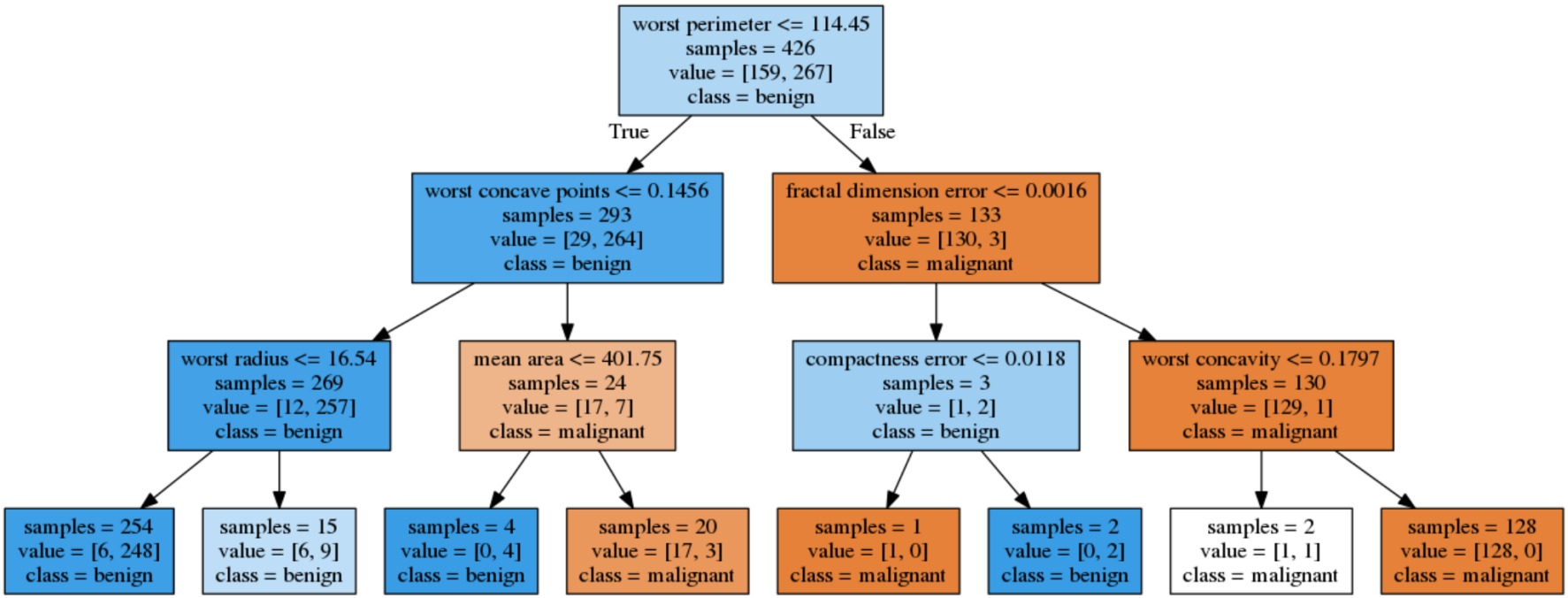
```
Max Depth: 1
Training Accuracy: 0.92
Testing Accuracy: 0.90

Max Depth: 2
Training Accuracy: 0.95
Testing Accuracy: 0.92

Max Depth: 3
Training Accuracy: 0.96
Testing Accuracy: 0.92

Max Depth: 4
Training Accuracy: 0.98
Testing Accuracy: 0.91
```

Decision Tree of Depth 3



https://necromuralist.github.io/machine-learning-studies/posts/decision-tree-classification/

Decision Tree of Depth 3

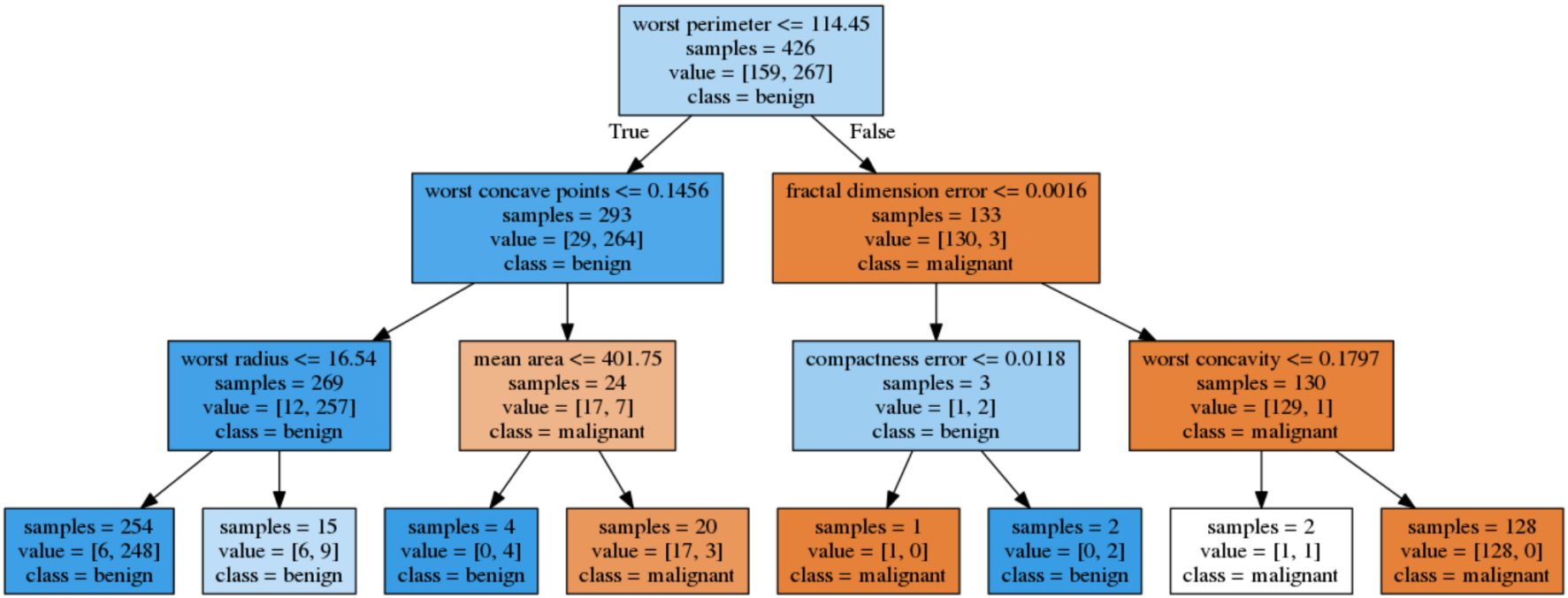


https://necromuralist.github.io/machine-learning-studies/posts/decision-tree-classification/

Decision Tree yields accurate and **explainable** results!

Decision Tree of Depth 3

Question: Can you find the training error of this classification tree?



https://necromuralist.github.io/machine-learning-studies/posts/decision-tree-classification/

Decision Tree yields accurate and **explainable** results!

References

A. Blum, J. Hopcroft and R. Kannan
Foundations of Data Science
https://www.cs.cornell.edu/jeh/book.pdf

G. James, D. Witten, T. Hastie, and R. Tibshirani.
An Introduction to Statistical Learning.
Springer, 2014. http://www-bcf.usc.edu/~gareth/ISL/

J. Leskovec, A. Rajaraman, J. D. Ullmann
Mining of Massive Datasets.
3rd edition, Cambridge University Press, 2020.

S. Shalev-Shwartz and S. Ben-David
Understanding Machine Learning: From Theory to Algorithms
Cambridge University Press, 2014.
https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/
understanding-machine-learning-theory-algorithms.pdf