ensemble
- bagging
- boosting

CS 412

APR 2ND – DECISION TREES

Final Project

Team picked Project Topic Next Friday 4/10

Official write up/description

Groups of up to 3

- Groups of 2, I expect ~50% more
- Groups of 3, ~100% more

Every group will need to meet digitally with me (~15 minutes) between 4/13 and 4/17 to make sure the project is feasible

Project will be 5-10 pages, single-spaced, but with figures

- Problem introduction
- Previous work
- Novel attempt
- Results
- Future work/Conclusion

Due: May 8th

Rationale for Ensemble Learning

No Free Lunch thm: There is no algorithm that is always the most accurate

Generate a group of base-learners which when combined have higher accuracy

Different learners use different
- Algorithms
- Parameters
- Representations (Modalities)
- Training sets
- Subproblems

combining "weak" base-learners

bagging: lowers variance

boosting: lowers bias

Boosting

Boosting works differently.

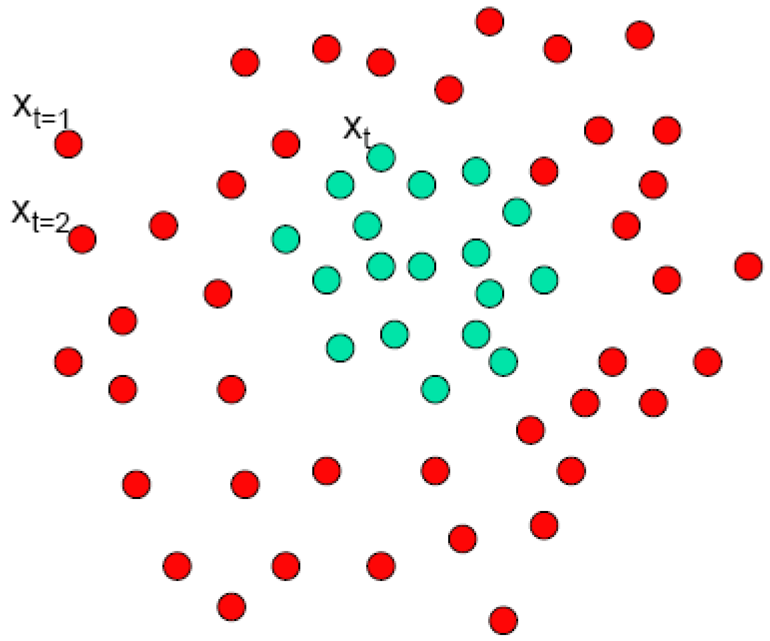
1. Boosting does not involve bootstrap sampling
2. Decision models are grown sequentially: each model is created using information from previously grown trees
3. Like bagging, boosting involves combining a large number of models, $f^1, \ldots, f^B$

Changing weights of incorrectly classified points

assign a weight to the model h_t

Boosting - Example

Weak classifier
- linear discriminator



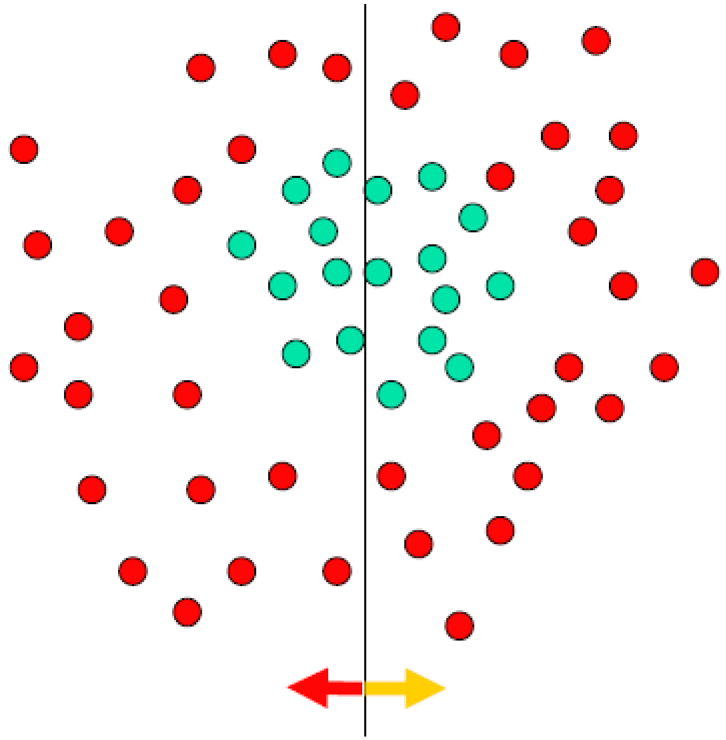
Each data point has a class label:

$$y_t = \begin{cases} +1 & (\text{red}) \\ -1 & (\text{green}) \end{cases}$$

and a weight:

$w_t = 1$

Boosting - Example



Each data point has

a class label:

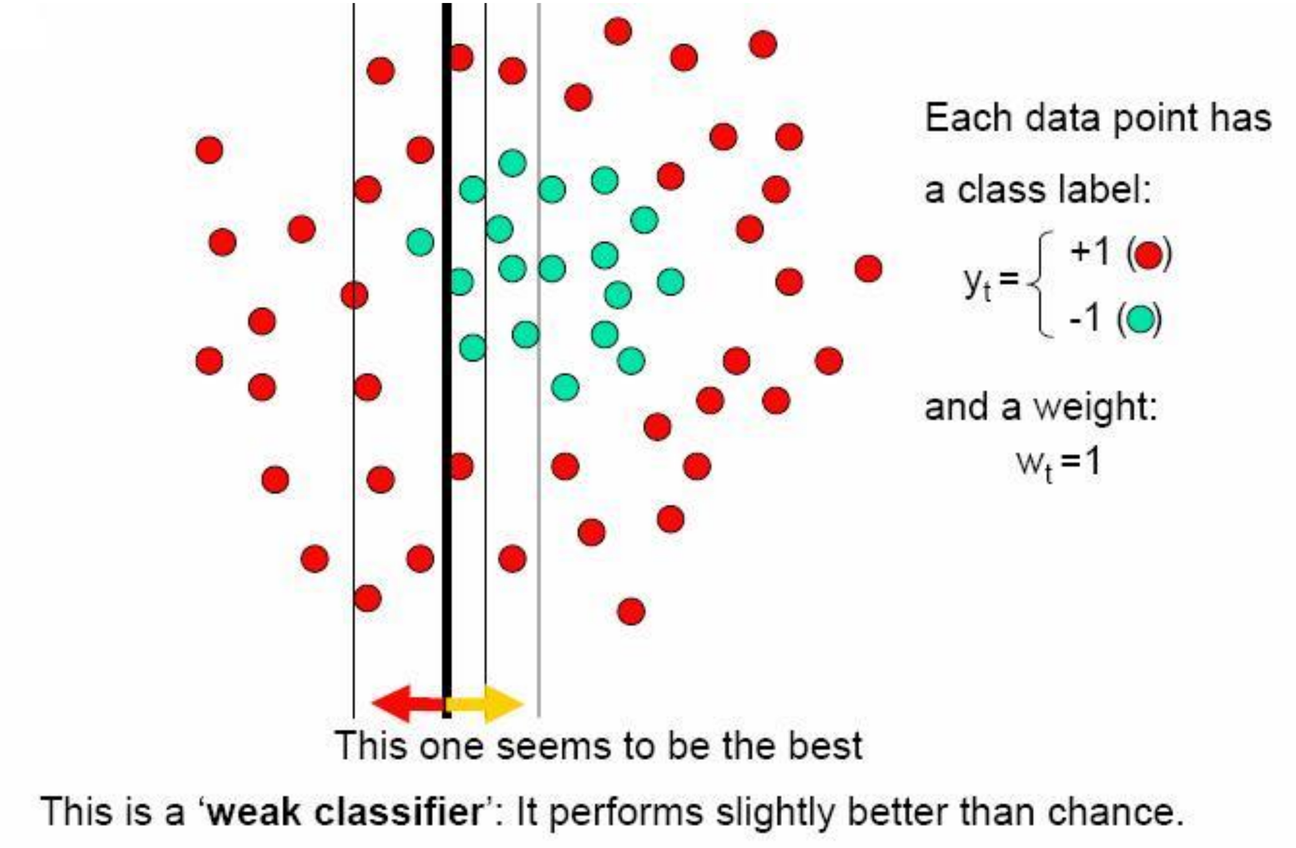
$$y_t = \begin{cases} +1 \ (\bullet) \\ -1 \ (\bullet) \end{cases}$$

and a weight:

$w_t = 1$

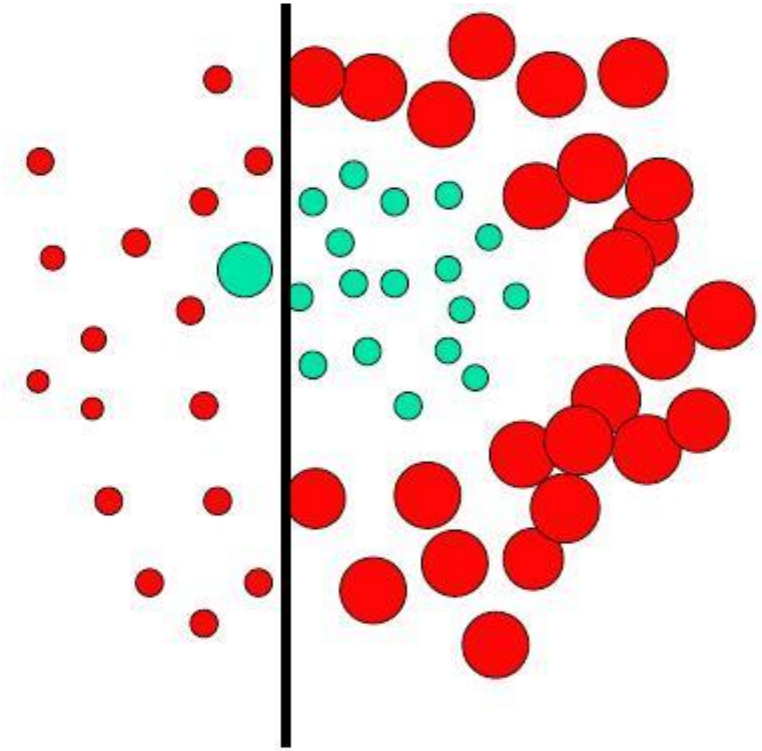
h => p(error) = 0.5 it is at chance

Boosting - Example



Boosting - Example

$\forall_i \; D_i = \frac{1}{n}$



Each data point has

a class label:

$$y_t = \begin{cases} +1 & (\bullet) \\ -1 & (\bullet) \end{cases}$$

$\epsilon_j = \sum \frac{1}{n}$ incorrect class

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

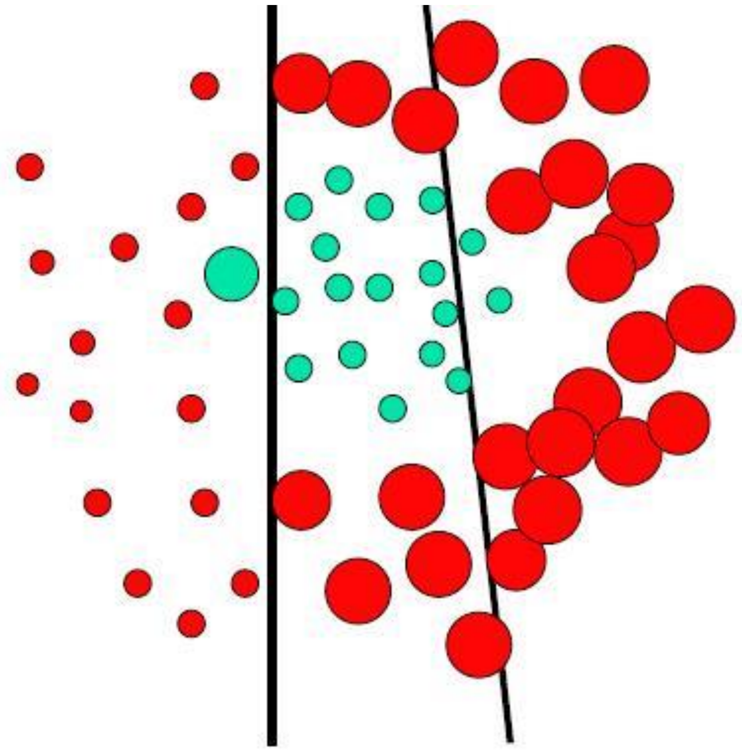
reweight them all

$\sum D_t = 1$

We set a new problem for which the previous weak classifier performs at chance again

h_2

Boosting - Example



Each data point has

a class label:

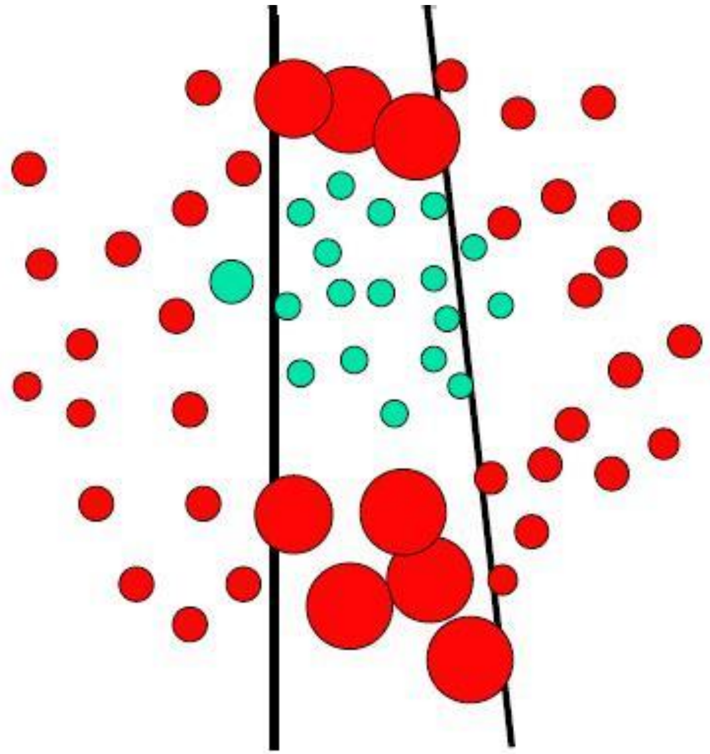
$$y_t = \begin{cases} +1 \ (\bullet) \\ -1 \ (\bullet) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Boosting - Example



Each data point has

a class label:

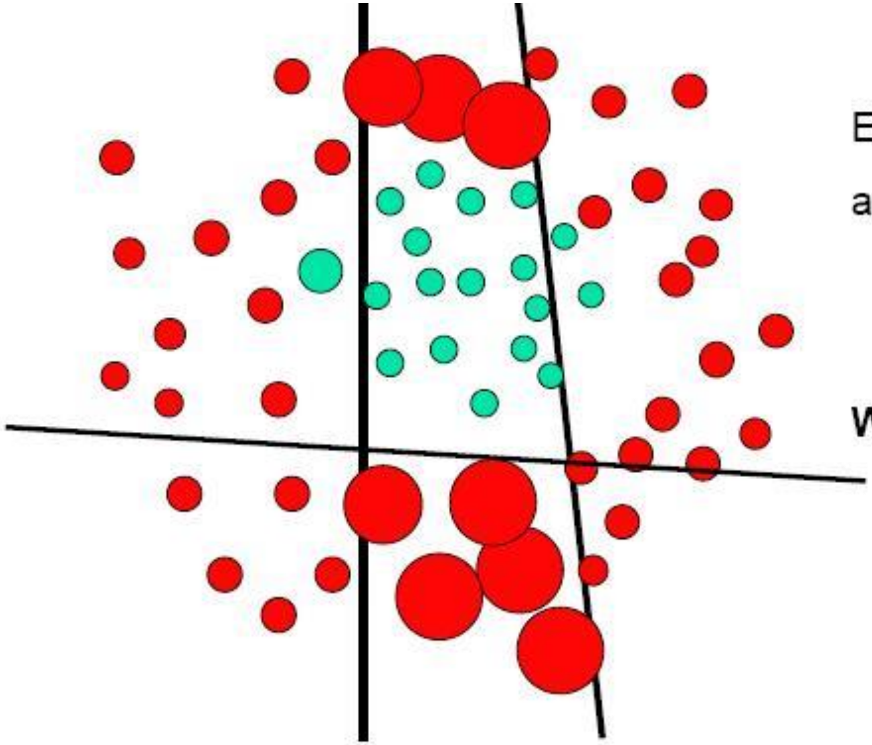
$$y_t = \begin{cases} +1 \\ -1 \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Boosting - Example



Each data point has

a class label:

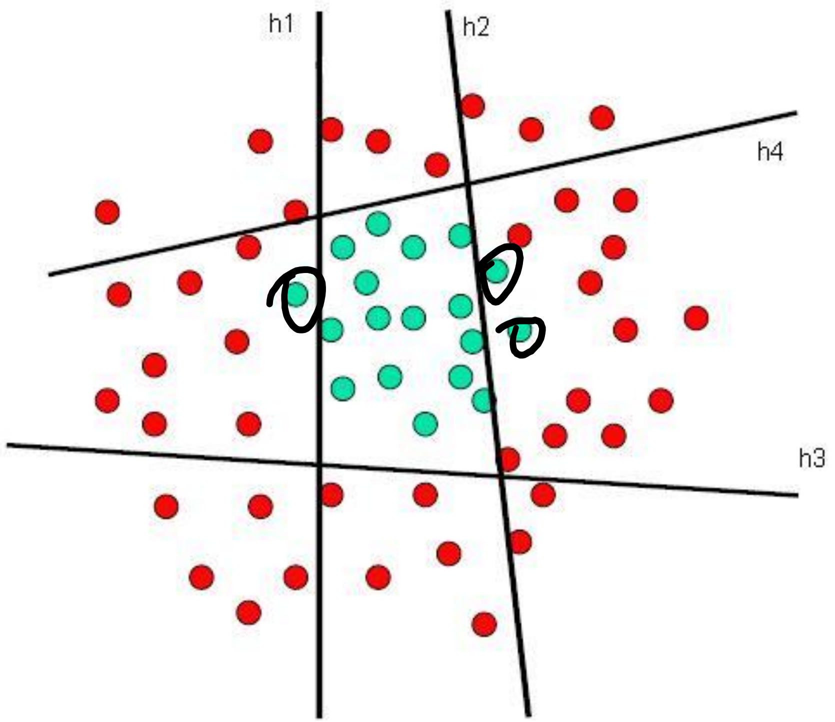
$$y_t = \begin{cases} +1 \; (\text{red}) \\ -1 \; (\text{green}) \end{cases}$$

We update the weights:

$$w_t \leftarrow w_t \exp\{-y_t H_t\}$$

We set a new problem for which the previous weak classifier performs at chance again

Boosting - Example



The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

iterations given as a tuning parameter

pick optimum value by cross-validation

Boosting

blue positive

$\alpha_i = \frac{(1-\varepsilon_i)}{\varepsilon_i}$

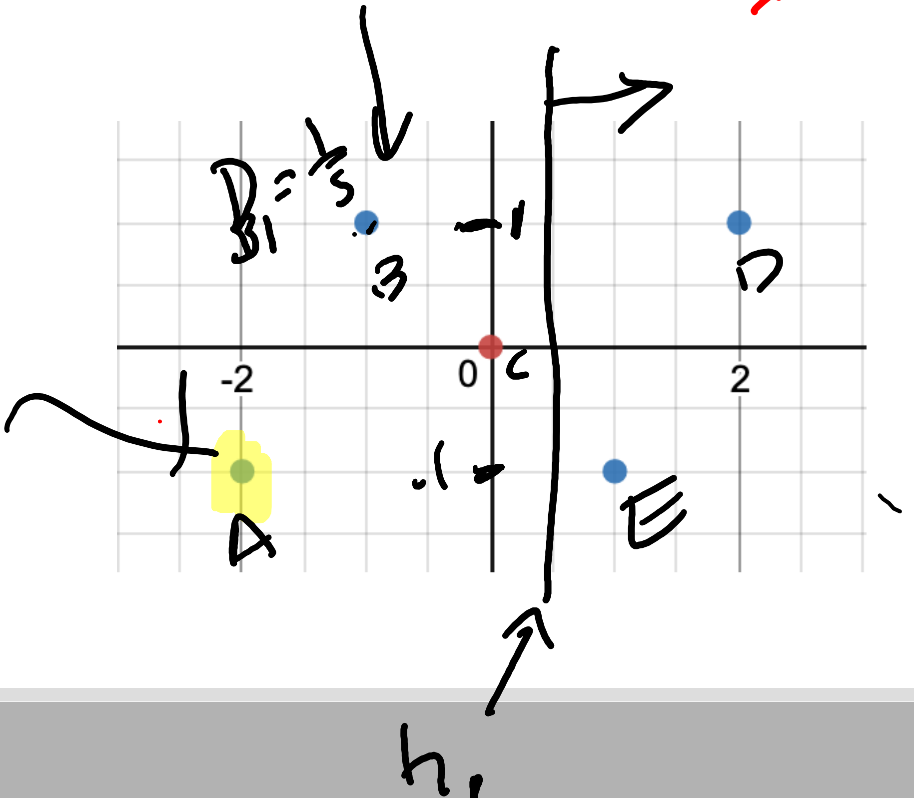
Perform 3 iterations of the boosting algorithm on the following dataset. Double the error for incorrect points at each iteration and weight each model with $\underbrace{\alpha_i = (1\text{-}\varepsilon)/\varepsilon}$

$D_{B2} = \boxed{2/7}$

$\varepsilon_1 = \sum_{i \in E} D_i = 2/5$

i ∈ E incorrectly classified

$\alpha_1 = \frac{3/5}{2/5} = 3/2$



Boosting

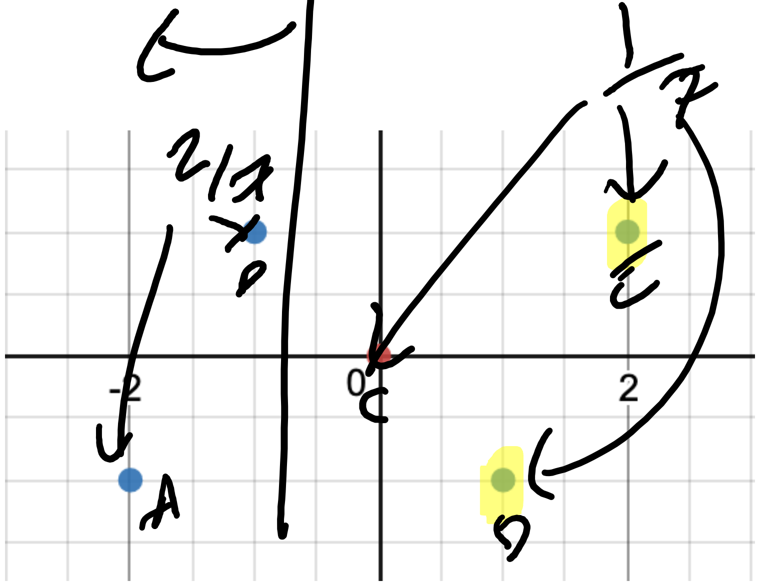
$$\sum \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3 =$$

Perform 3 iterations of the boosting algorithm on the following dataset. Double the error for incorrect points at each iteration and weight each model with $\alpha_i = (1-\varepsilon)/\varepsilon$

$$\varepsilon_2 = \sum D_i = \underset{D}{\frac{1}{7}} + \underset{E}{\frac{1}{7}} = 2/7$$

$$\alpha_2 = \frac{5/7}{2/7} = \frac{5}{2}$$

h_2



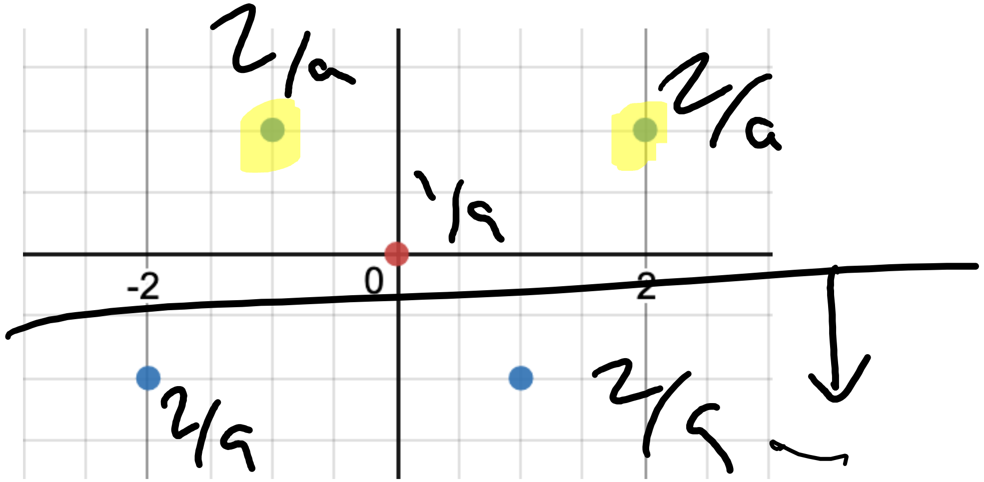
New weights @ t=3 $\quad \frac{2}{79} + \frac{2}{79} + \frac{1}{79} + \frac{2}{79} + \frac{2}{79} = \frac{9}{7}$

Boosting

Perform 3 iterations of the boosting algorithm on the following dataset. Double the error for incorrect points at each iteration and weight each model with $\alpha_i = (1-\varepsilon)/\varepsilon$

$$\varepsilon_3 = \frac{2}{9} + \frac{2}{9} = \frac{4}{9}$$

$$\alpha_3 = \frac{5/9}{4/9} = \frac{5}{4}$$



Boosting

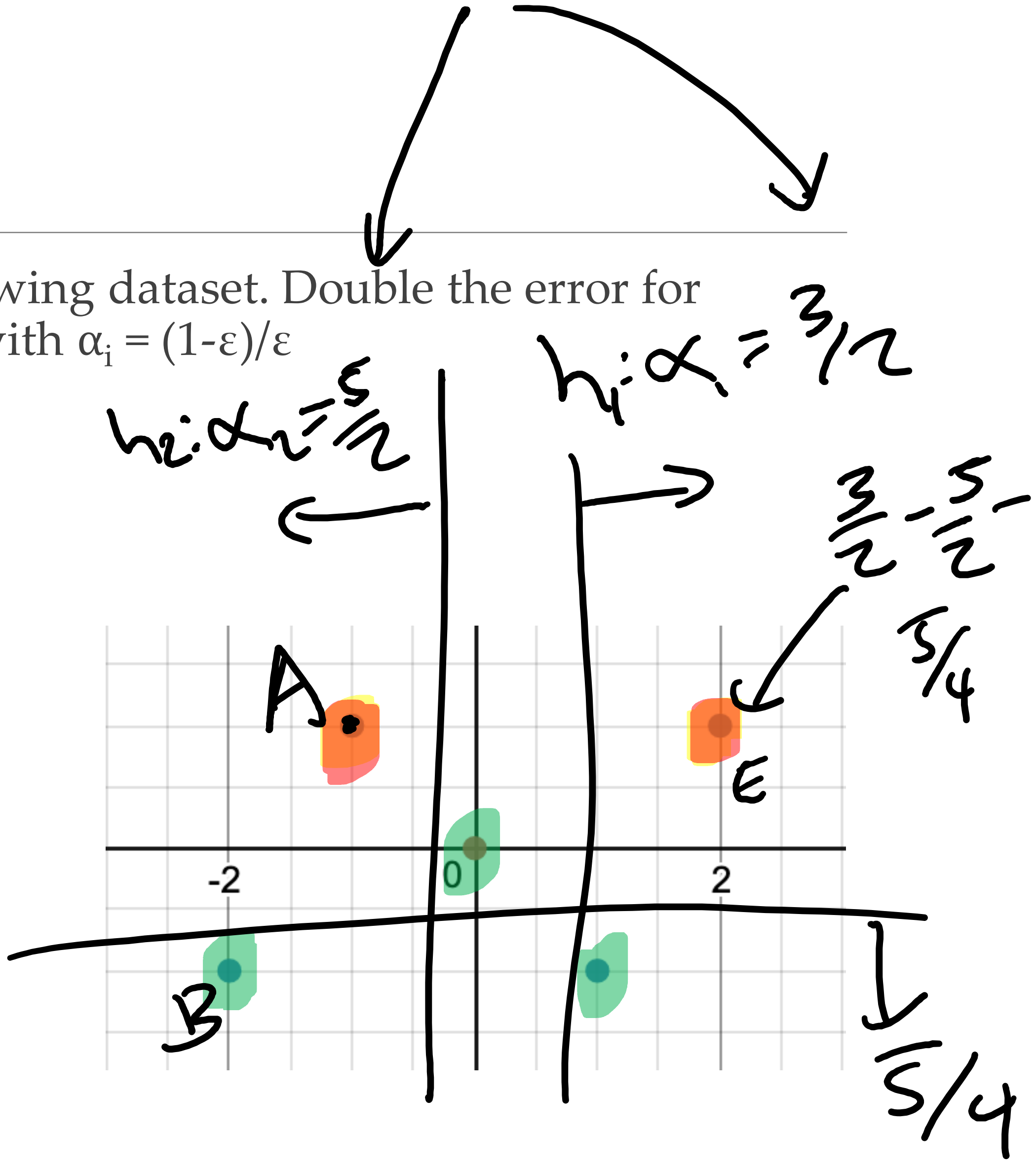
Perform 3 iterations of the boosting algorithm on the following dataset. Double the error for incorrect points at each iteration and weight each model with $\alpha_i = (1-\varepsilon)/\varepsilon$

Class A: h_1: $-\frac{3}{2}$ $\rangle$ $\frac{2}{2}$

h_2: $\frac{5}{2}$

h_3: $-\frac{5}{4}$

ε: $\frac{-1}{4}$



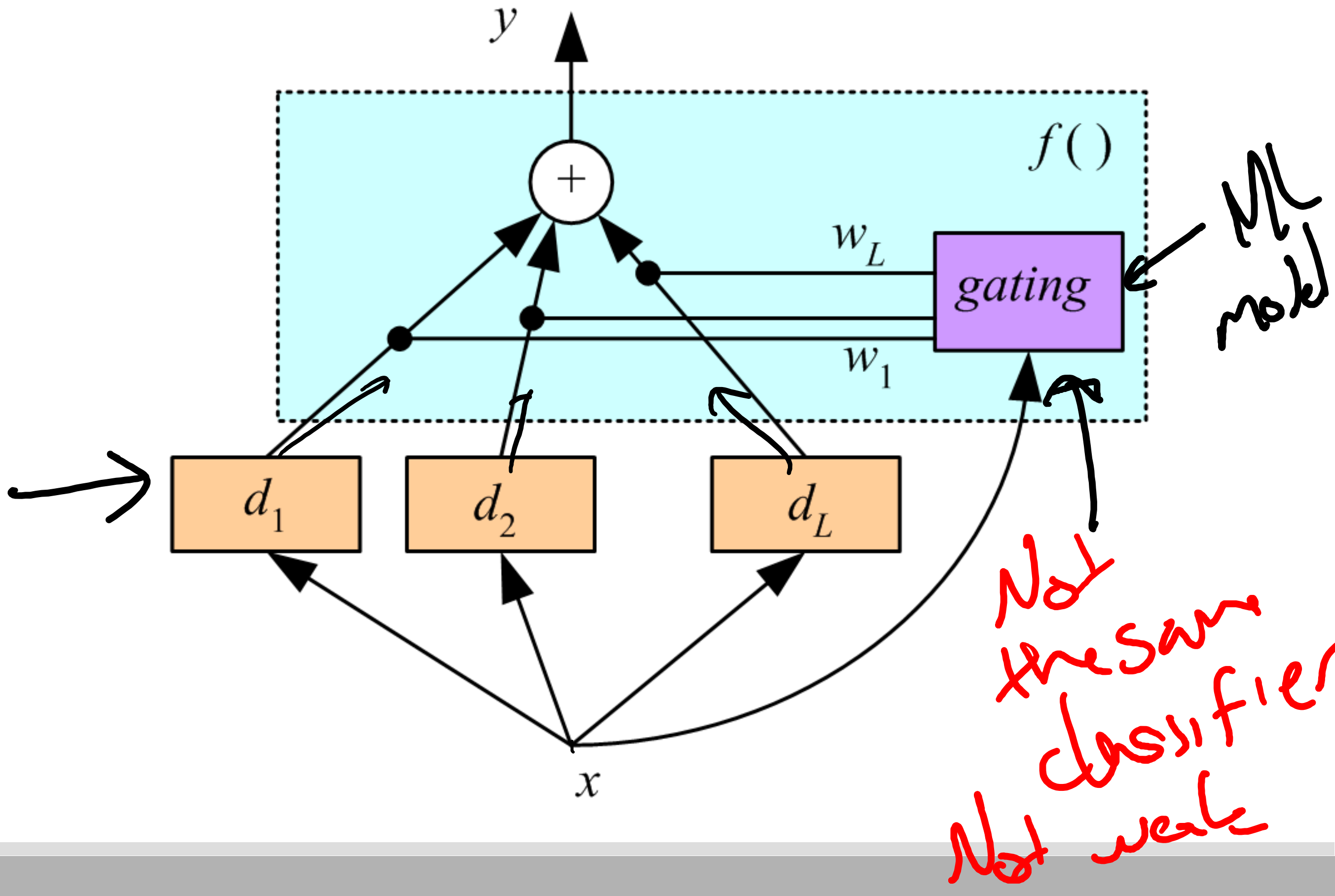
Mixture of Experts: Gating

Voting where weights are input-dependent (gating)

$$y = \sum_{j=1}^{L} w_j(x) d_j$$

(Jacobs et al., 1991)

Experts or gating can be nonlinear

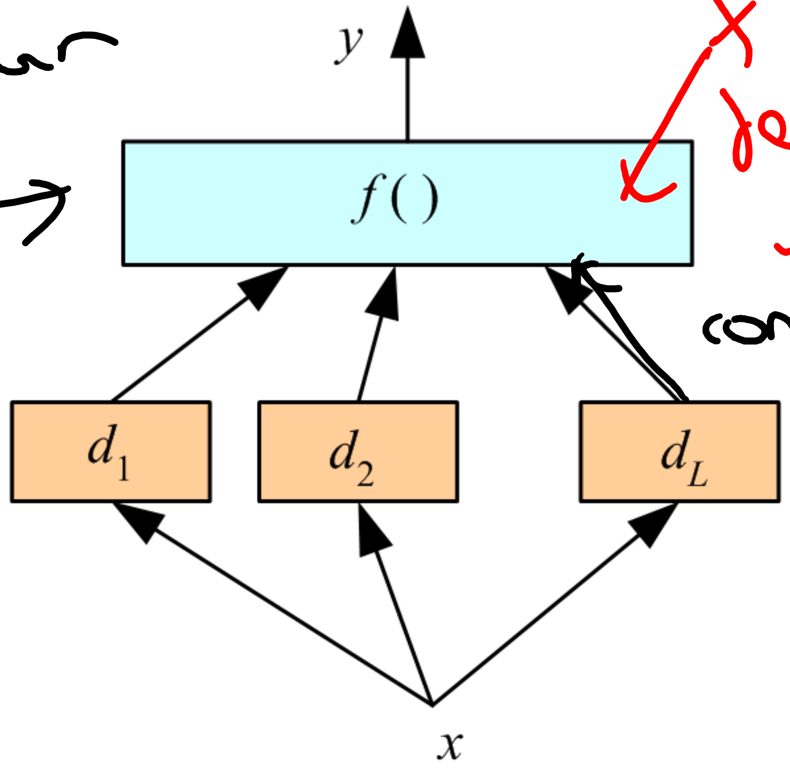


Stacking

Combiner f () is another learner (Wolpert, 1992) that corrects the bias of base learners

Combiner f should be trained on data unused in training the base learners

rather than voting



f learns dependencies w/in learners

confidence

high confidence for d_i might impact accuracy for d_k

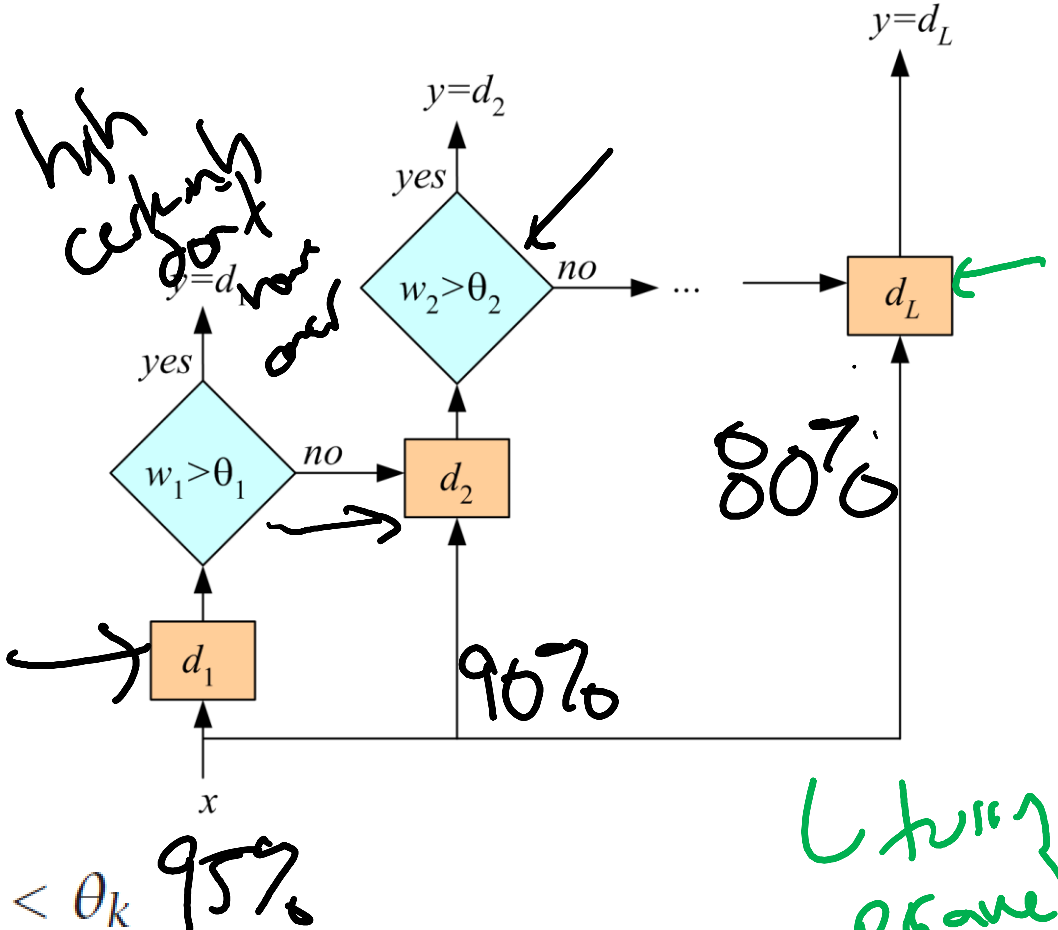
for cancer

Cascading

Cascade learners in order of complexity

Use dj only if preceding ones are not confident (wi)

asymmetric error



$$1/K < \theta_j \le \theta_{j+1} < 1$$

$$y_i = d_{ji} \text{ if } w_j > \theta_j \text{ and } \forall k < j, w_k < \theta_k$$

tuning parameter

Fine-Tuning an Ensemble

Given an ensemble of dependent classifiers, do not use it as is, try to get independence

bias reduction

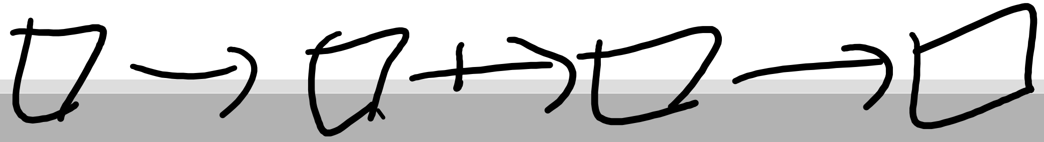
- Inaccurate base learners can worsen accuracy (think of majority voting)

not too weak 60-70%

Subset selection: Forward (growing)/Backward (pruning) approaches to improve accuracy/diversity/independence

give each model a different subset of the features

can also select a subset of the learners

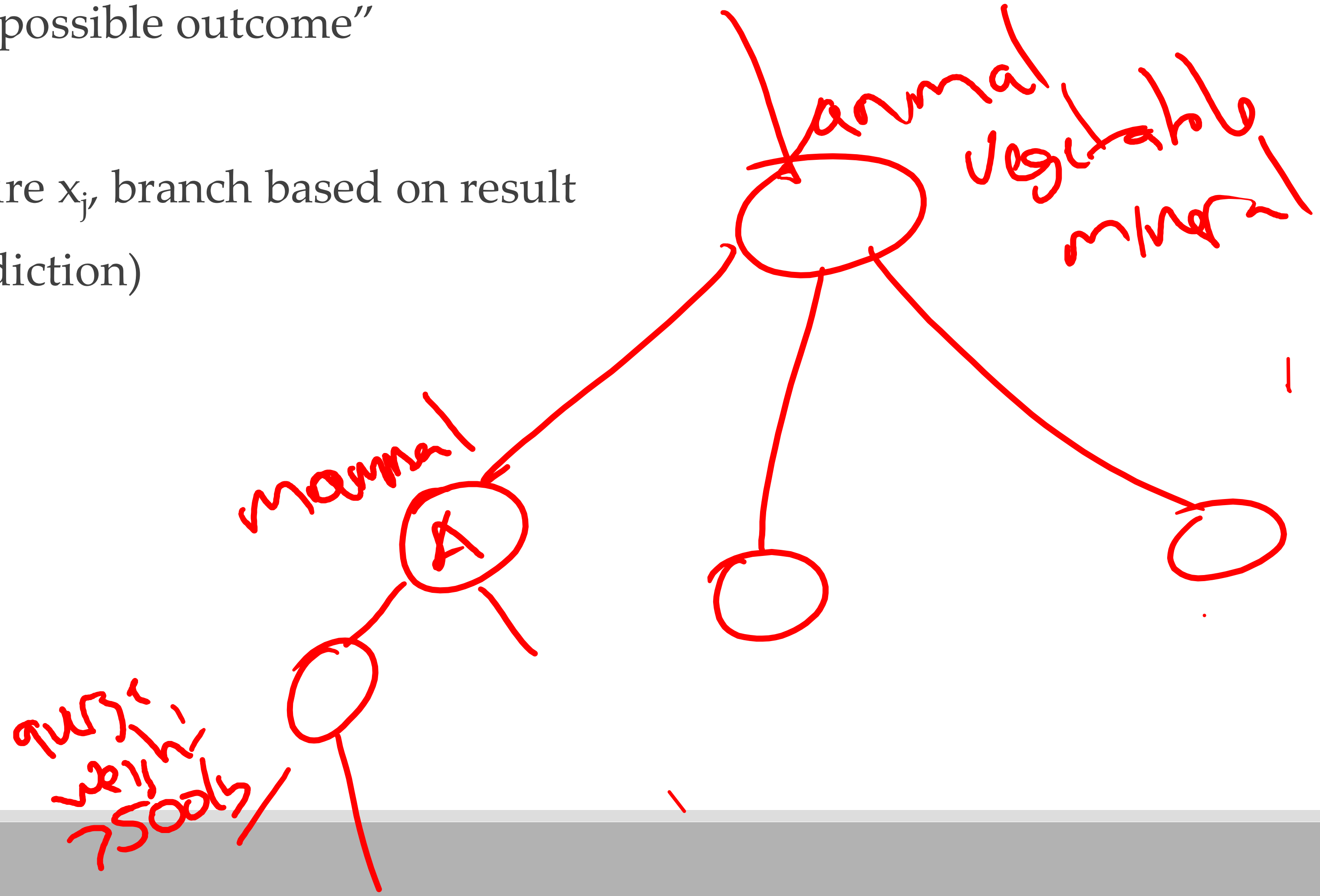


Decision Trees

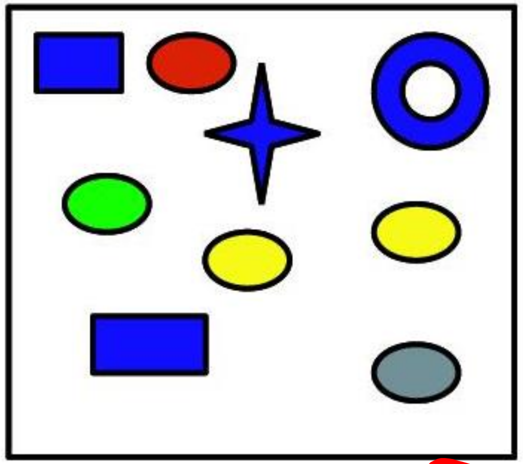
“20 questions game for each possible outcome”

Nodes: test the value of feature x_j, branch based on result

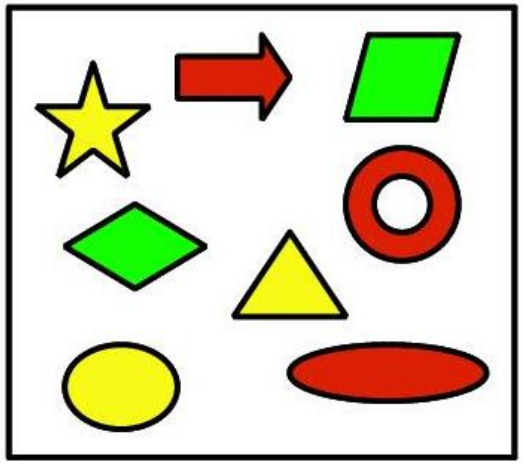
Leafs: provide the class (prediction)



yes



no



Color

b y g r

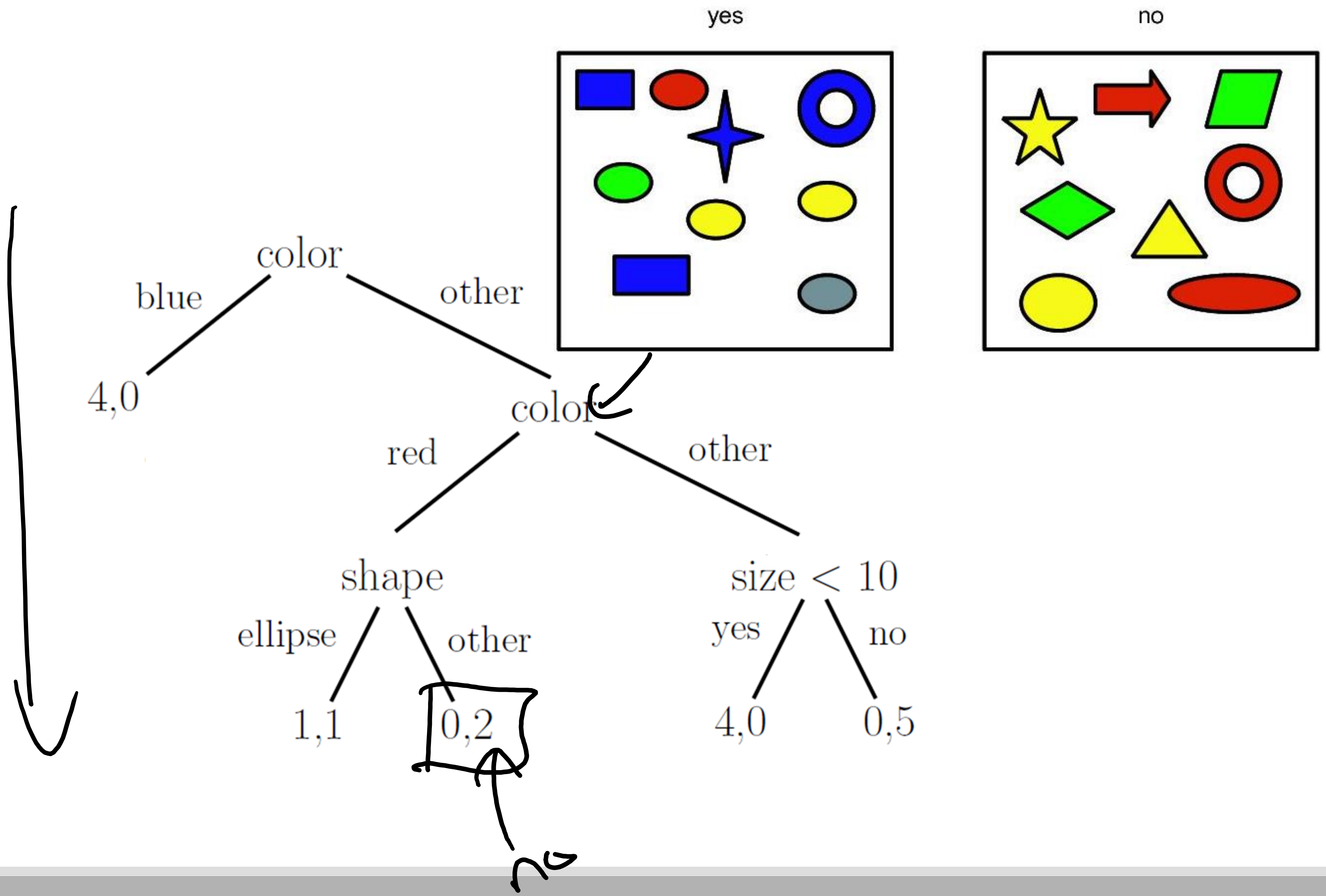
B

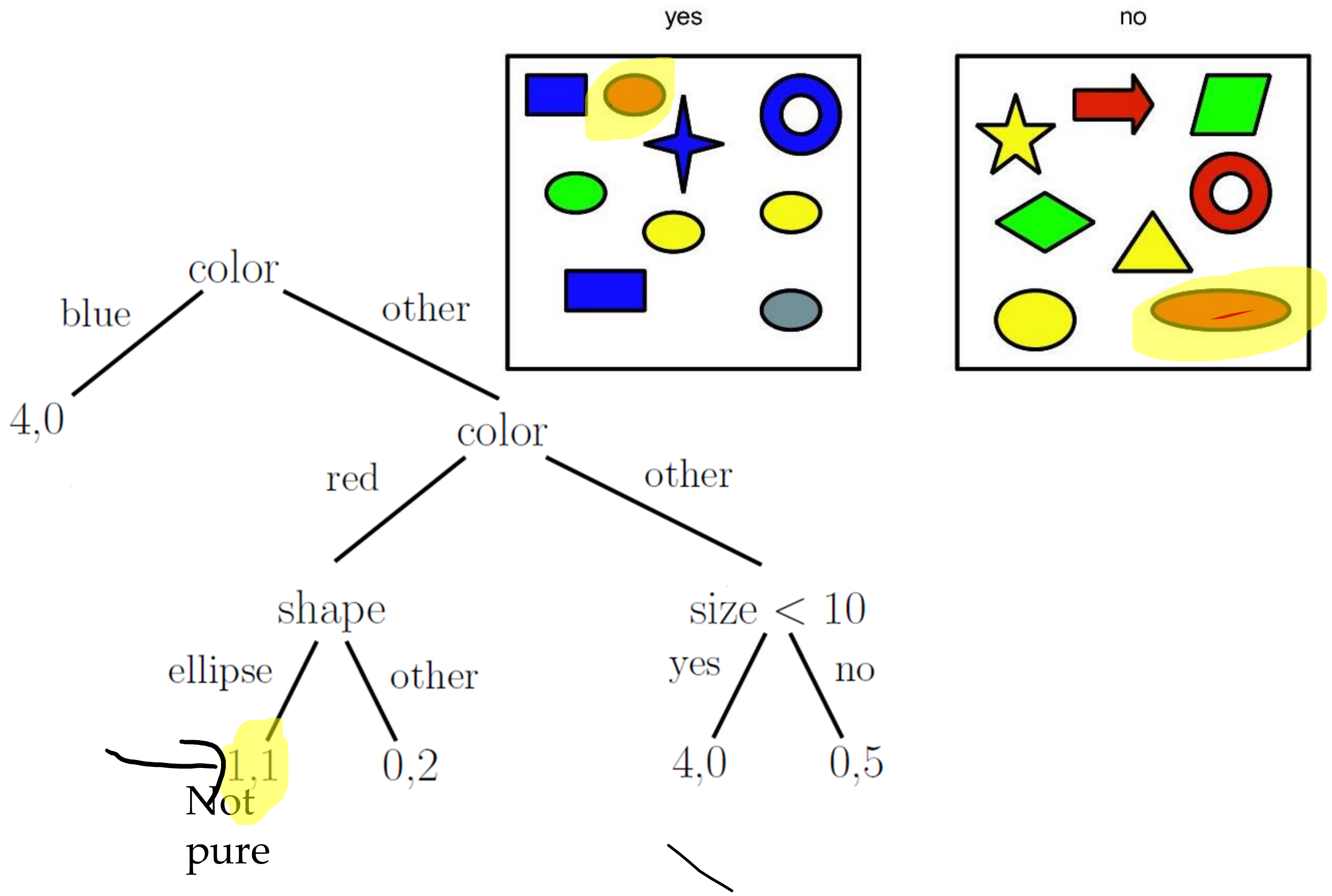
4

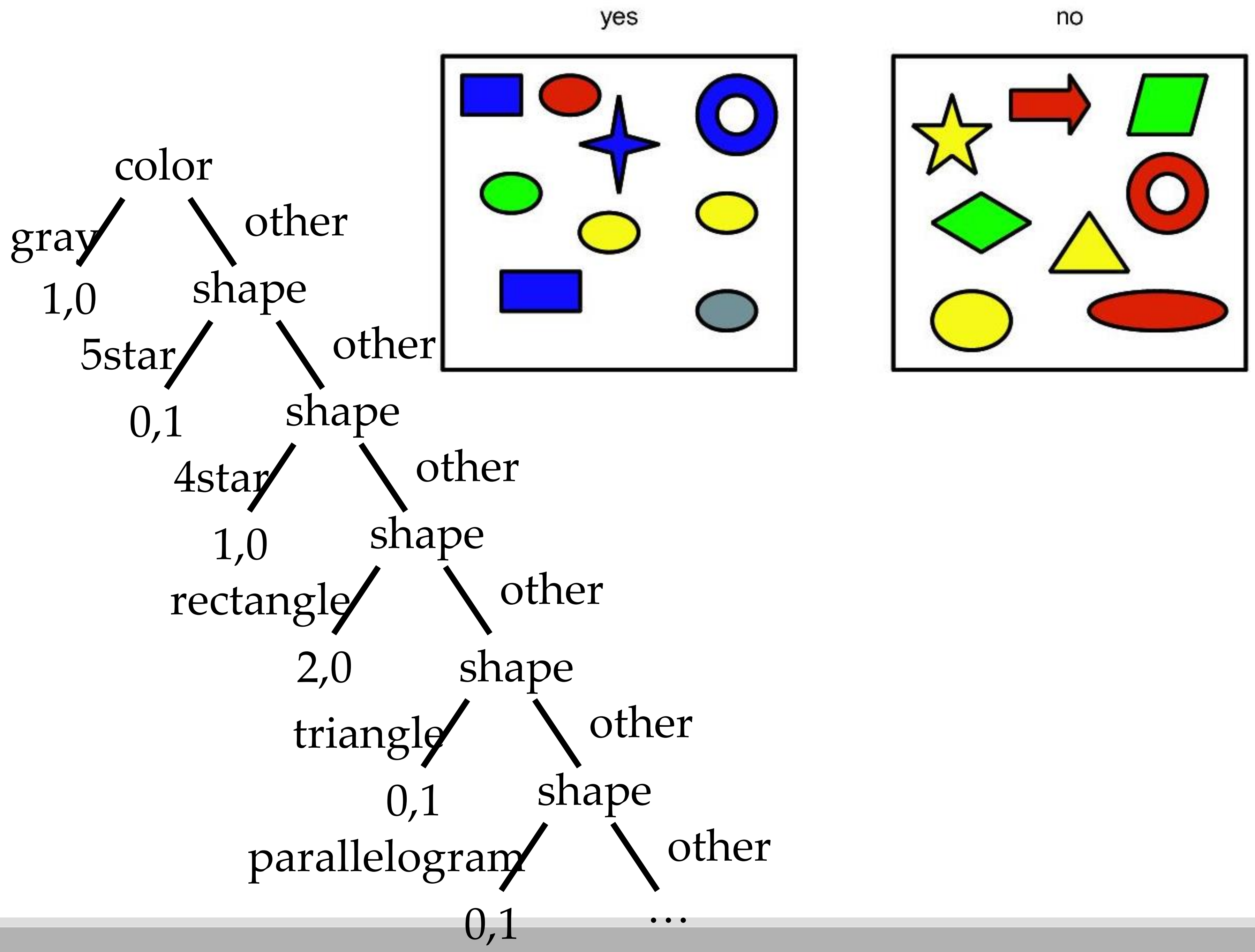
Or

3n

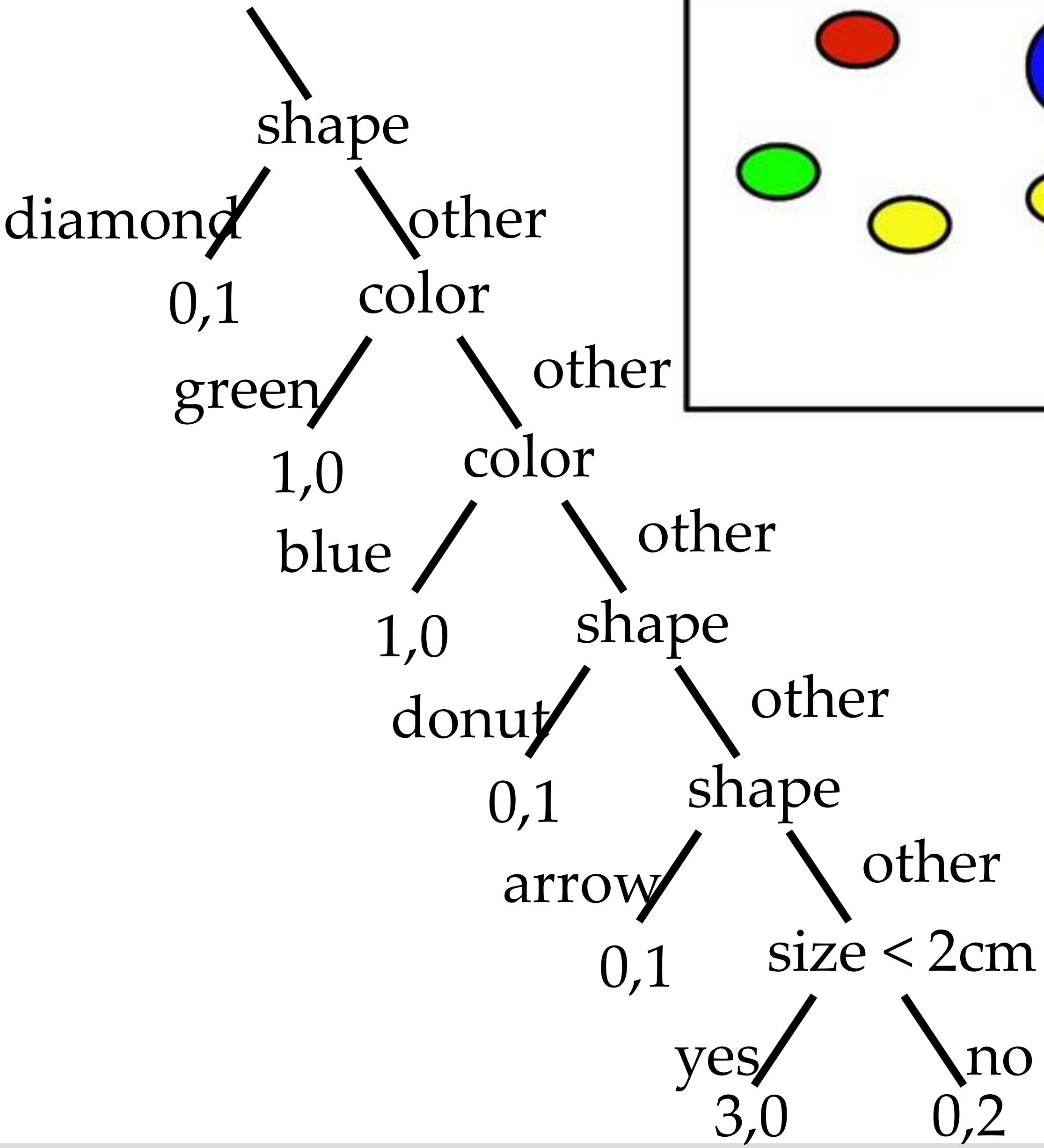
4

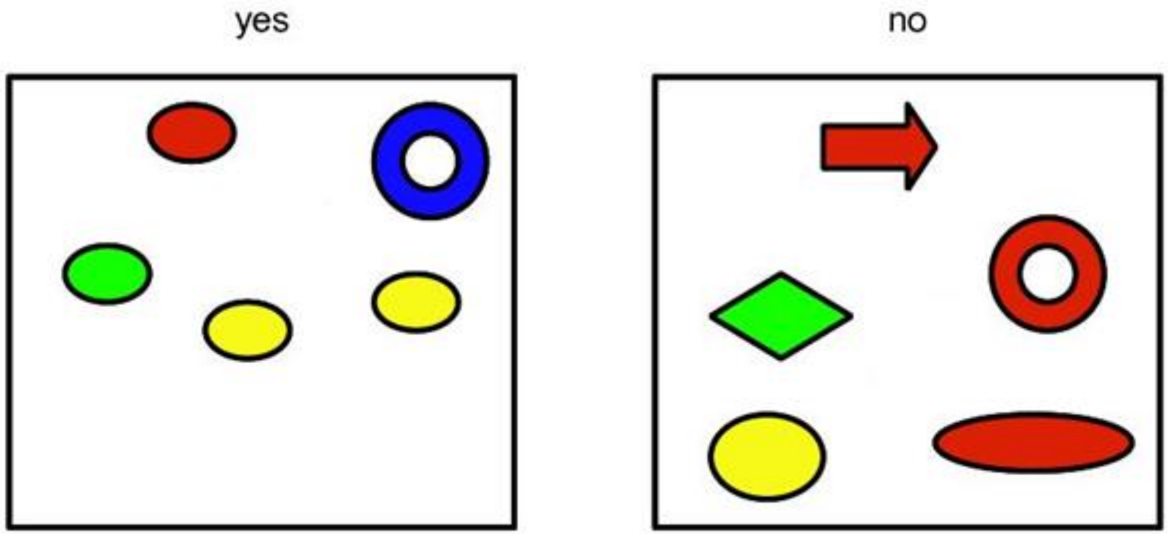

yes
no
color
blue
other
4,0
color
red
other
shape
size < 10
ellipse
other
yes
no
1,1
0,2
4,0
0,5
no

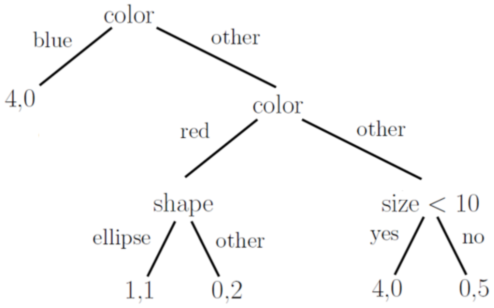

yes
no
color
blue
other
4,0
color
red
other
shape
size < 10
ellipse
other
yes
no
1,1
0,2
4,0
0,5
Not
pure



yes
no
color
gray
other
1,0
shape
5star
other
0,1
shape
4star
other
1,0
shape
rectangle
other
2,0
shape
triangle
other
0,1
shape
parallelogram
other
0,1
…


shape
diamond
0,1
other
color
green
1,0
other
color
blue
1,0
other
shape
donut
0,1
other
shape
arrow
0,1
other
size < 2cm
yes
3,0
no
0,2

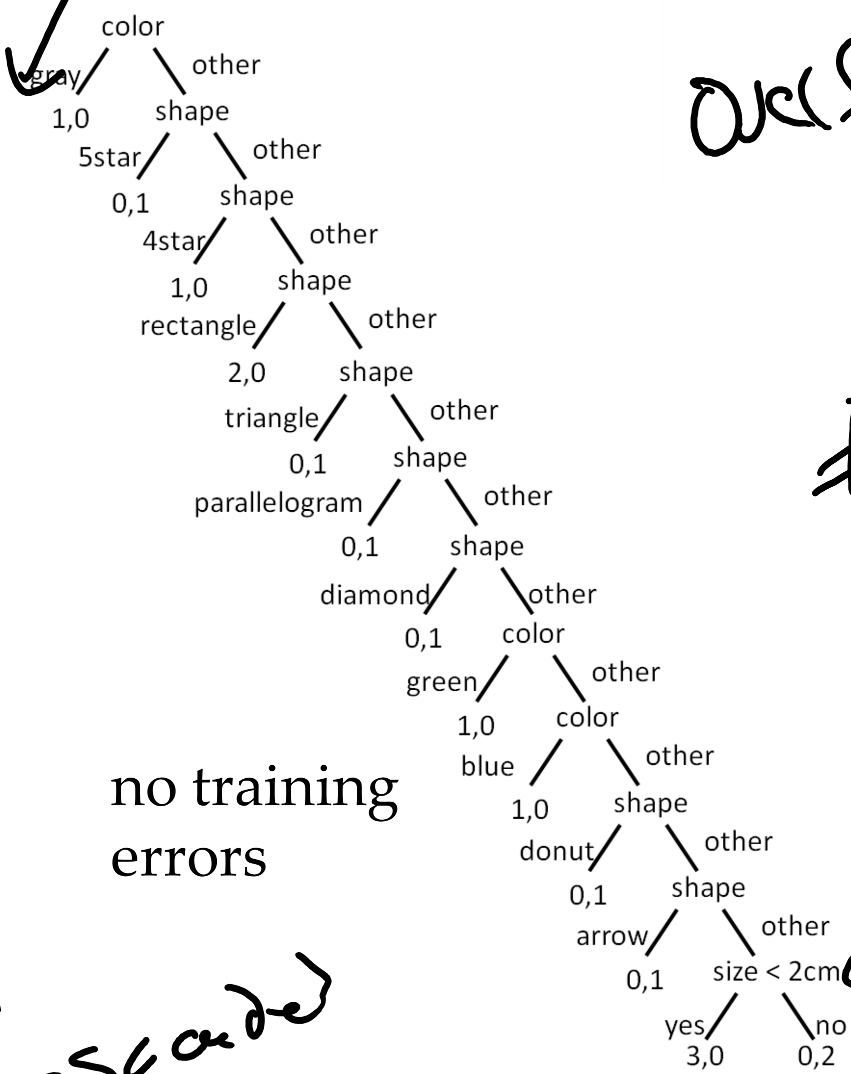

yes
no



1 training error

or

Overfitting



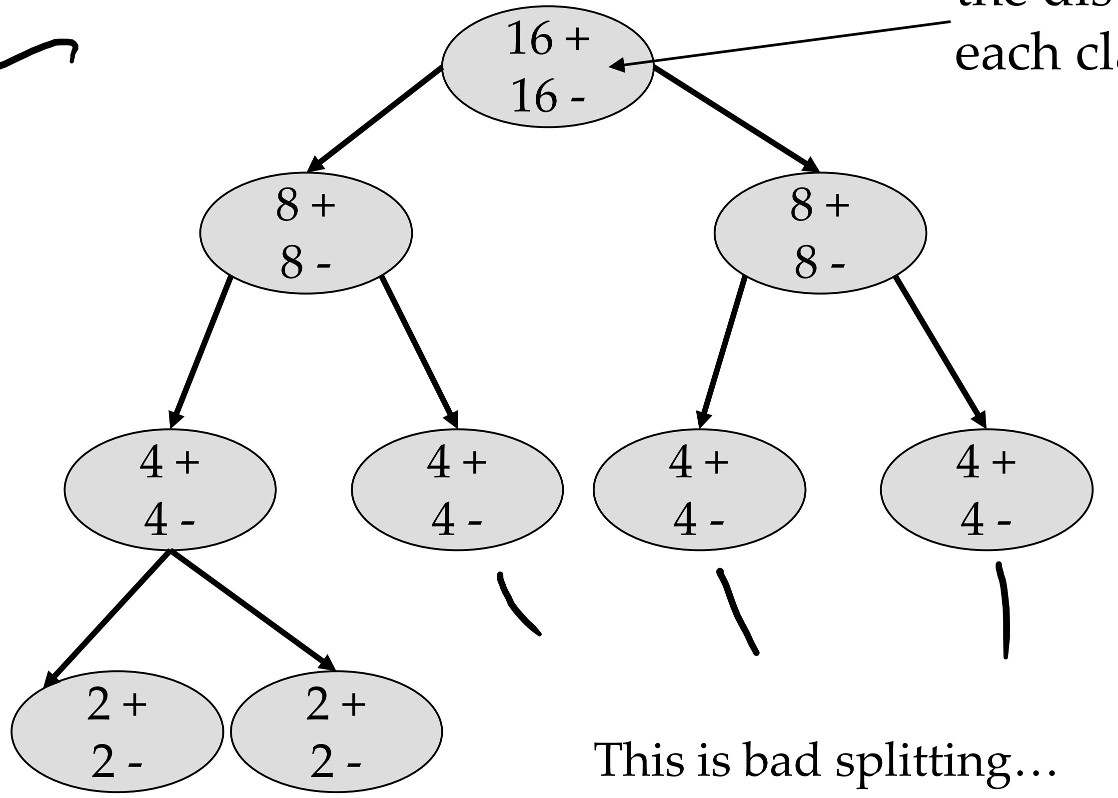
no training errors

\# decision nodes

Cascaded Decision stump = tree w/depth 1

Which attribute to select for splitting?

the distribution of
each class (not attribute)

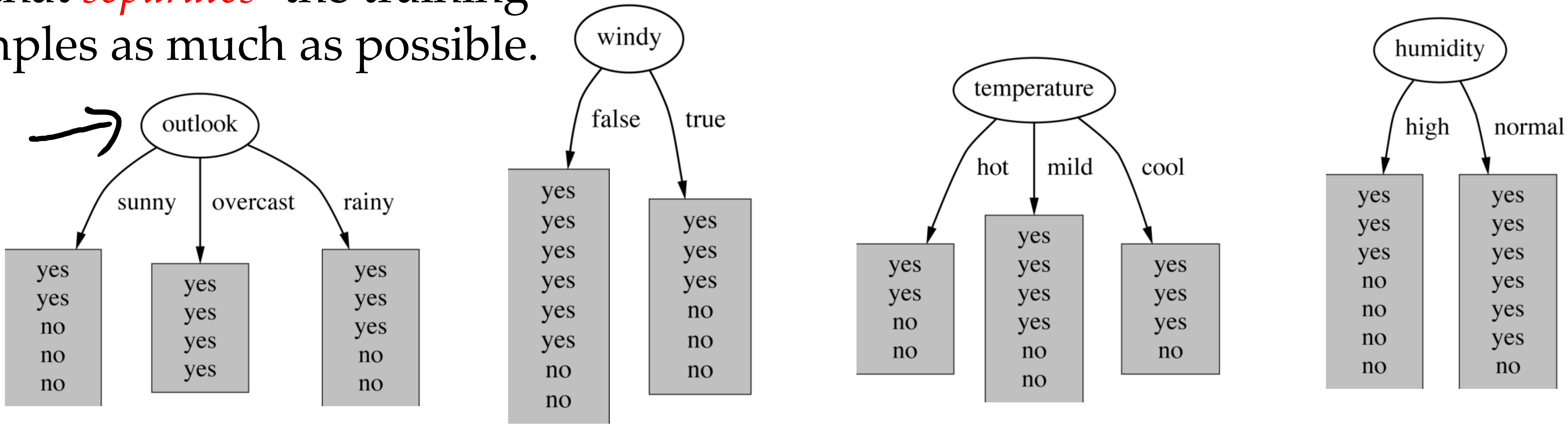


This is bad splitting…

How do we choose the test ?

Which attribute should be used as the test?

Intuitively, you would prefer the one that *separates* the training examples as much as possible.



Decision tree: divide and conquer

Supervised learning: classification and regression

Internal decision nodes implements a test function

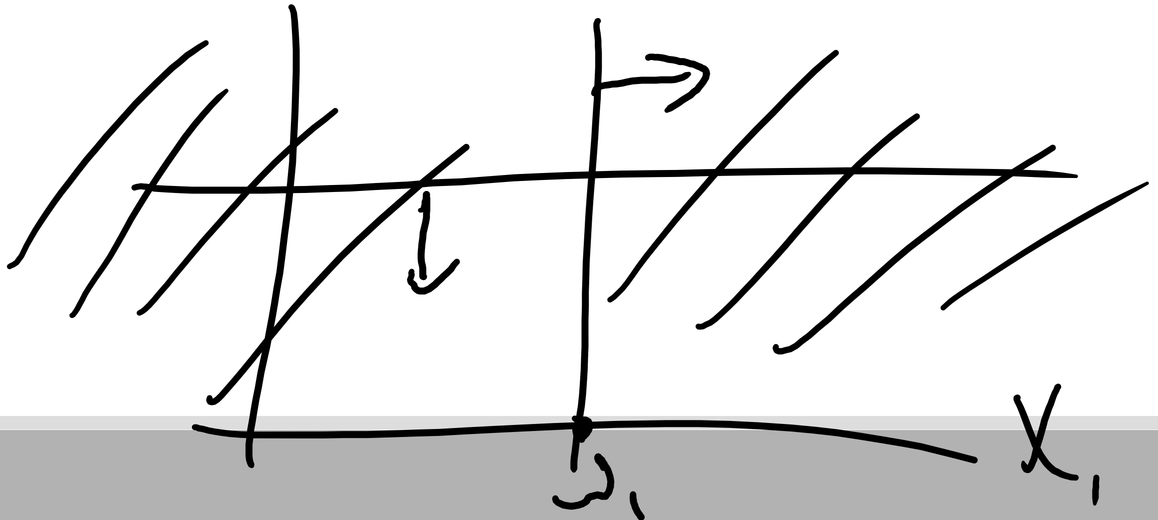
- Univariate: Uses a single attribute, x_i **– This is used most frequently**
  - Numeric x_i : Binary split : $x_i > w_m$
  - Discrete x_i : n-way split for n possible values or binary
- Multivariate: Uses multiple/all attributes, x

only ask a question about 1 feature at a time

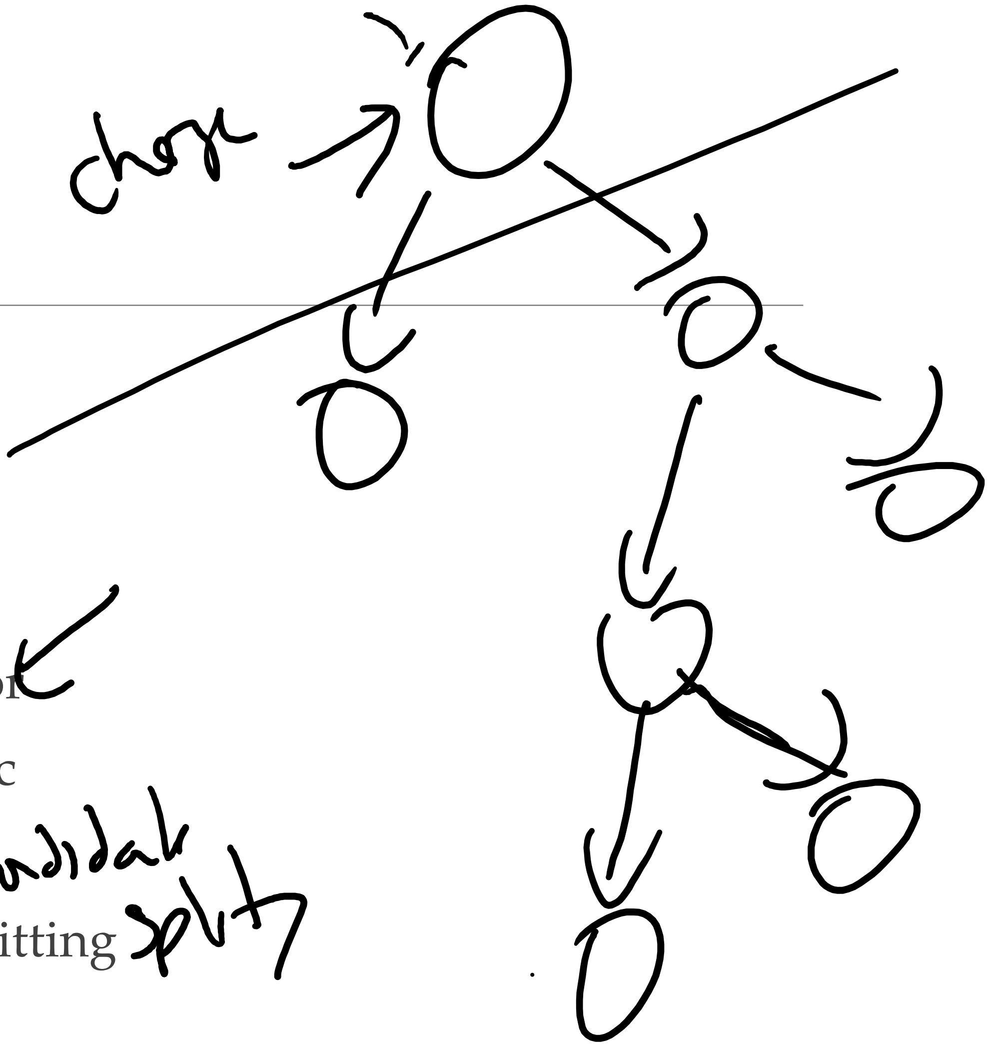
Leaves

- Classification: Class labels, or proportions
- Regression: Numeric; r average, or local fit

Highly interpretable



Classification Trees



Algorithms differ in branching models
- Pick a feature xj
- Discrete case (with n values): split into n branches
- Numeric case: discretize into two by thresholding
- fm(x) : xj > wm0 (threshold)

Goal: find the smallest tree that has low/zero training error

NP-complete, forced to use local search based on heuristic

Greedy: each step we look for the best split among candidate split

Score(D1, D2,… Dk) measuring "goodness" of splitting

the data into k subsets

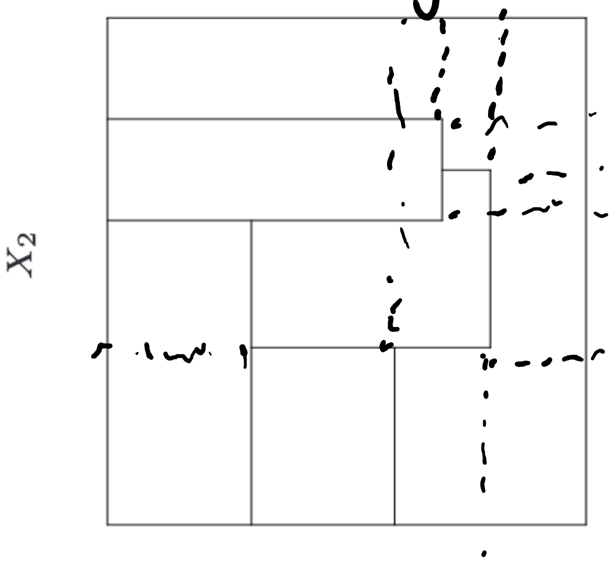
Continue recursively until no more split (leaf node)

purity thresholds

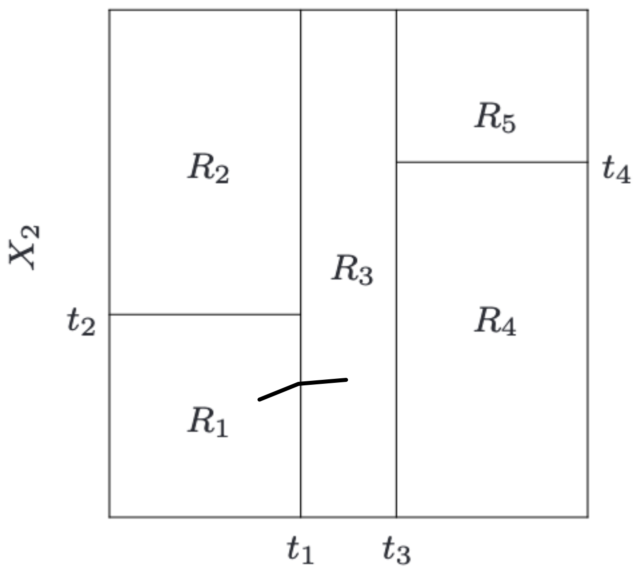
Region Types

What sorts of regions can the decision tree make?

could generate



a

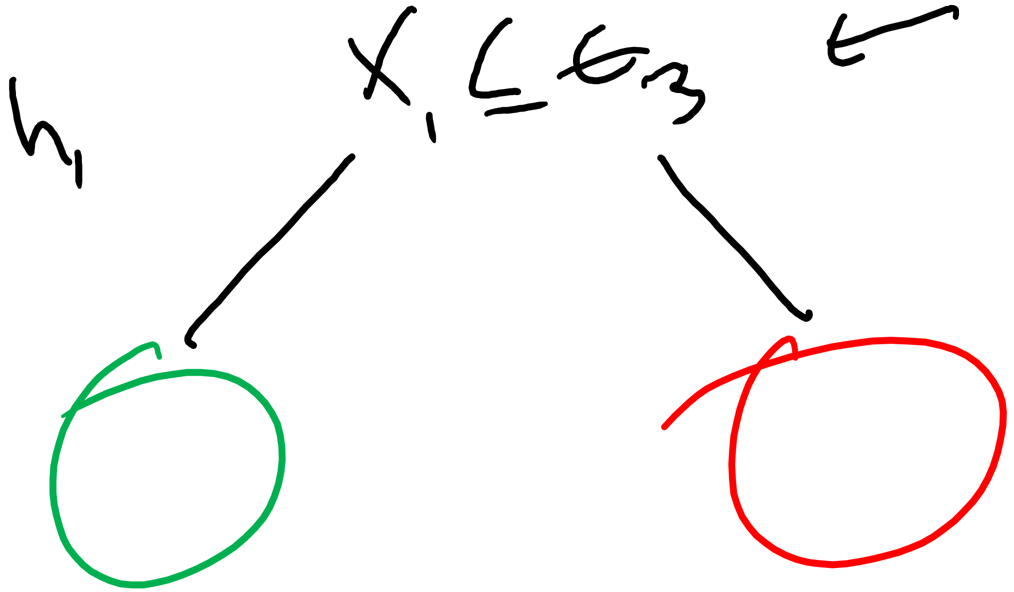


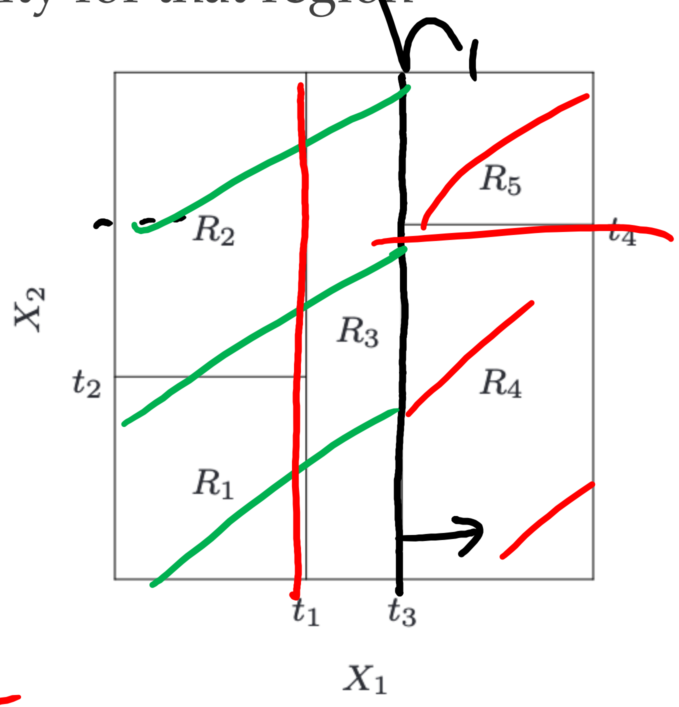
b

Region Types

What sorts of regions can the decision tree make?

Each "bin" is decided by whatever is the majority for that region



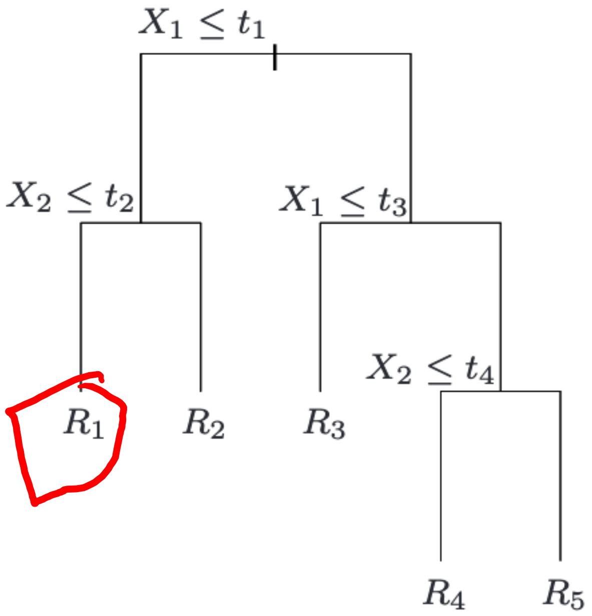


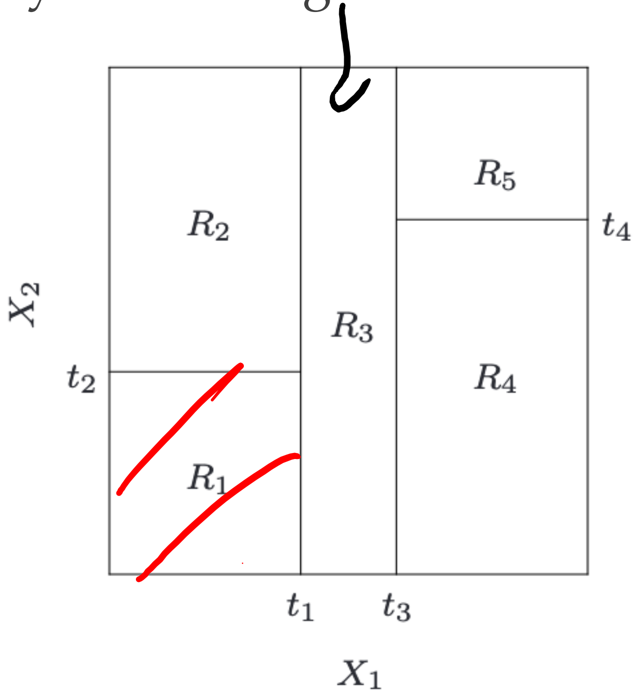
equivalent to DT
on next slide

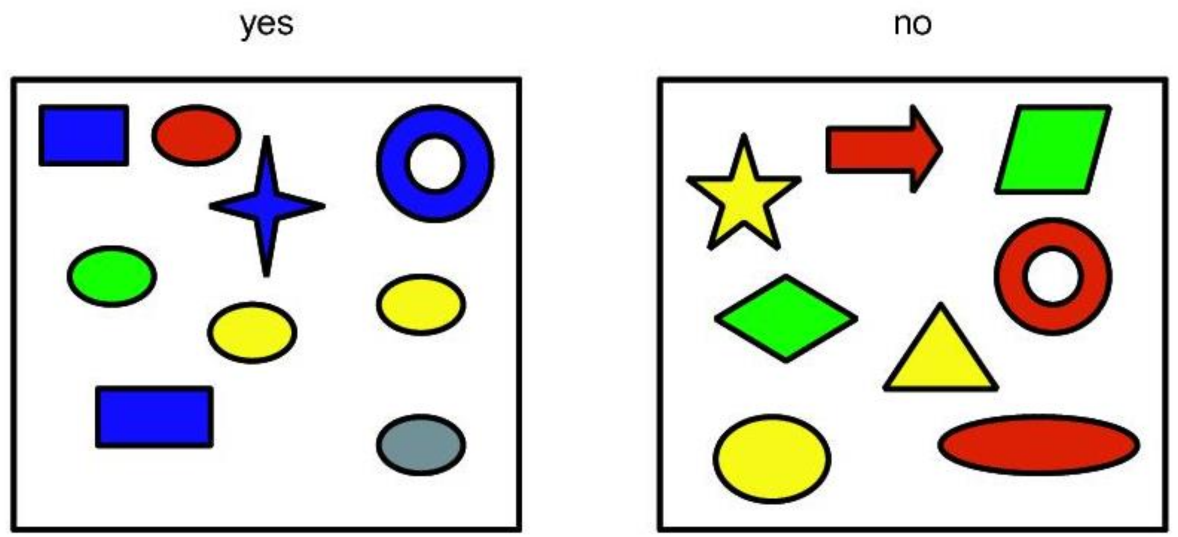
Region Types

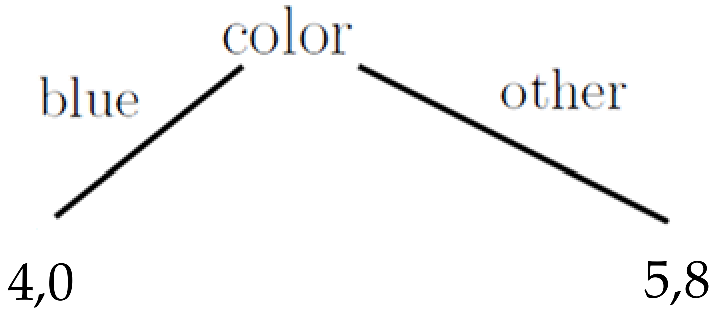
What sorts of regions can the decision tree make?

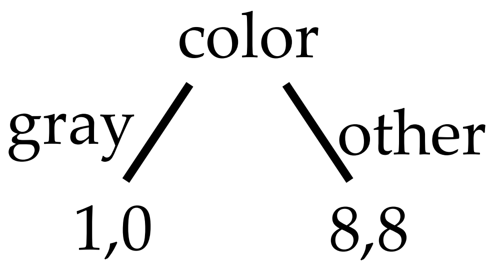
Each "bin" is decided by whatever is the majority for that region











Error rate: 5/17

Error rate: 8/17

Use majority label at the leaf, then compute error rate

Accuracy score pitfall

Accuracy score pitfall

Feature 1

400,200 400,200

Error rate: (200+200)/1200

Feature 2

250,240 550,160

Error rate: (240+160)/1200

Accuracy score pitfall

A

Feature 1

400,200 400,200

Error rate: (200+200)/1200

B

Feature 2

250,240 550,160

more purity

Error rate: (240+160)/1200

Both have the same error rate!!!

Which is "progressing more" towards a lower error?