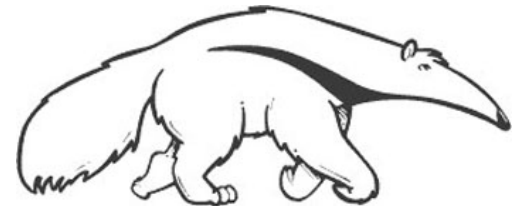


# Machine Learning Classifiers: Many Diverse Ways to Learn

Introduction to Artificial Intelligence  
Prof. Richard Lathrop

Read Beforehand: R&N 18.5-12, 20.2.2



# Outline

- Different types of learning problems
- Different types of learning algorithms
- Supervised learning
  - Decision trees
  - Naïve Bayes
  - Perceptrons, Multi-layer Neural Networks

## You will be expected to know

- Classifiers:
  - Decision trees
  - K-nearest neighbors
  - Naïve Bayes
  - Perceptrons, Support vector Machines (SVMs), Neural Networks
- Decision Boundaries for various classifiers
  - What can they represent conveniently? What not?

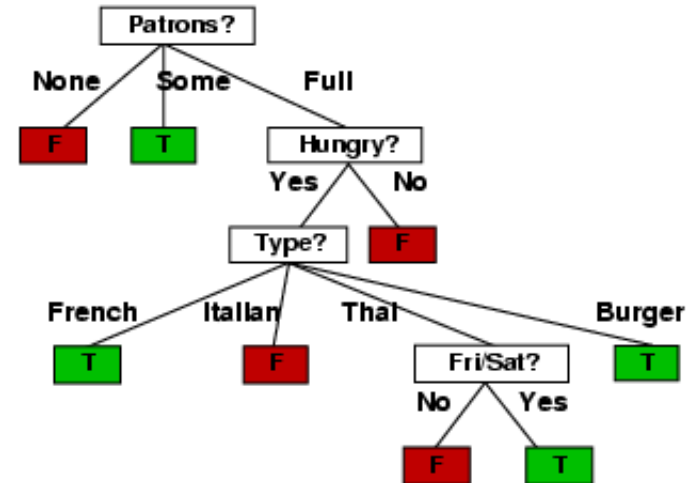
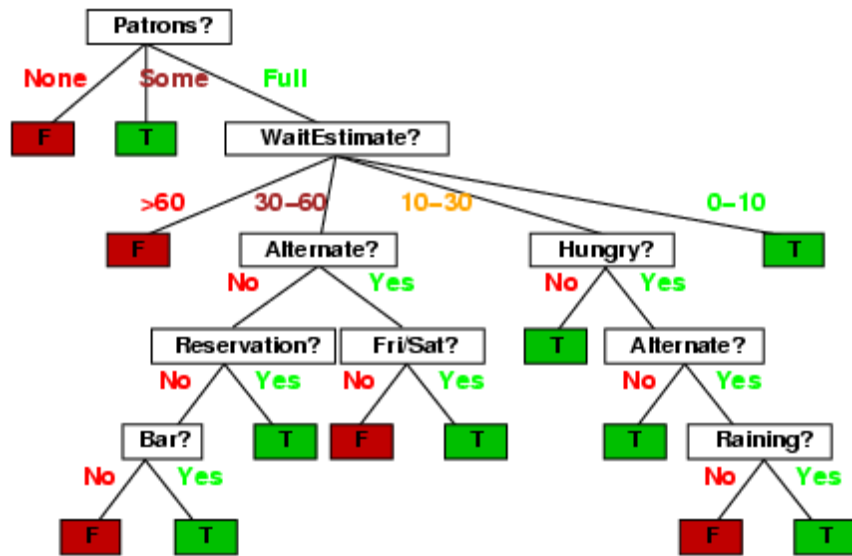
# Inductive learning

- Let  $\underline{x}$  represent the input vector of attributes
  - $x_j$  is the  $j$ th component of the vector  $x$
  - $x_j$  is the value of the  $j$ th attribute,  $j = 1, \dots, d$
- Let  $f(\underline{x})$  represent the value of the target variable for  $\underline{x}$ 
  - The implicit mapping from  $x$  to  $f(\underline{x})$  is unknown to us
  - We just have training data pairs,  $D = \{\underline{x}, f(\underline{x})\}$  available
- We want to learn a mapping from  $\underline{x}$  to  $f$ , i.e.,  
 $h(\underline{x}; \theta)$  is “close” to  $f(x)$  for all training data points  $\underline{x}$   
  
 $\theta$  are the parameters of our predictor  $h(..)$
- Examples:
  - $h(\underline{x}; \theta) = \text{sign}(w_1x_1 + w_2x_2 + w_3)$
  - $h_k(\underline{x}) = (x_1 \text{ OR } x_2) \text{ AND } (x_3 \text{ OR } \text{NOT}(x_4))$

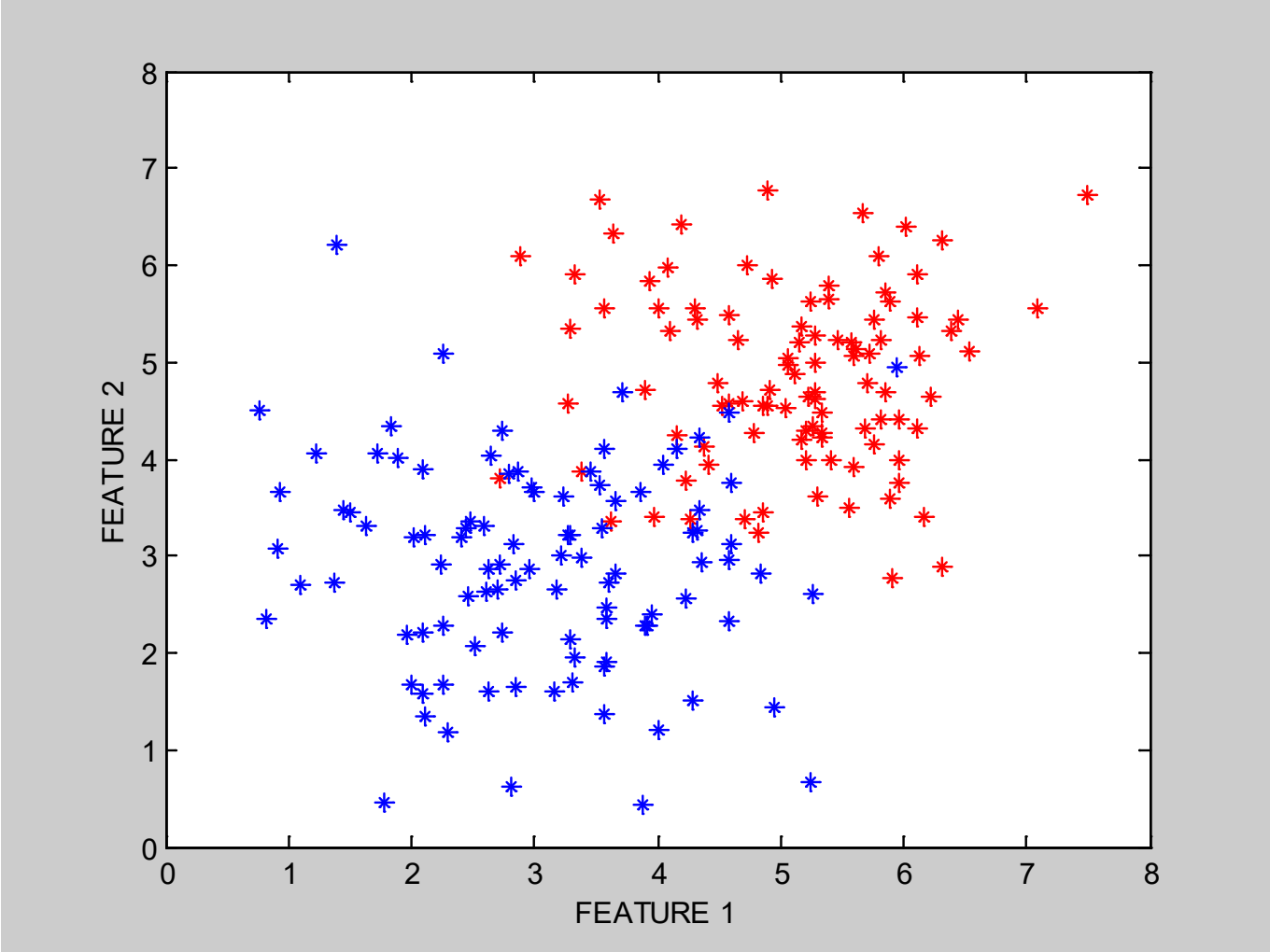
## Training Data for Supervised Learning

Example	Attributes										Target <i>Wait</i>
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	
$X_1$	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
$X_2$	T	F	F	T	Full	\$	F	F	Thai	30–60	F
$X_3$	F	T	F	F	Some	\$	F	F	Burger	0–10	T
$X_4$	T	F	T	T	Full	\$	F	F	Thai	10–30	T
$X_5$	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
$X_6$	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
$X_7$	F	T	F	F	None	\$	T	F	Burger	0–10	F
$X_8$	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
$X_9$	F	T	T	F	Full	\$	T	F	Burger	>60	F
$X_{10}$	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
$X_{11}$	F	F	F	F	None	\$	F	F	Thai	0–10	F
$X_{12}$	T	T	T	T	Full	\$	F	F	Burger	30–60	T

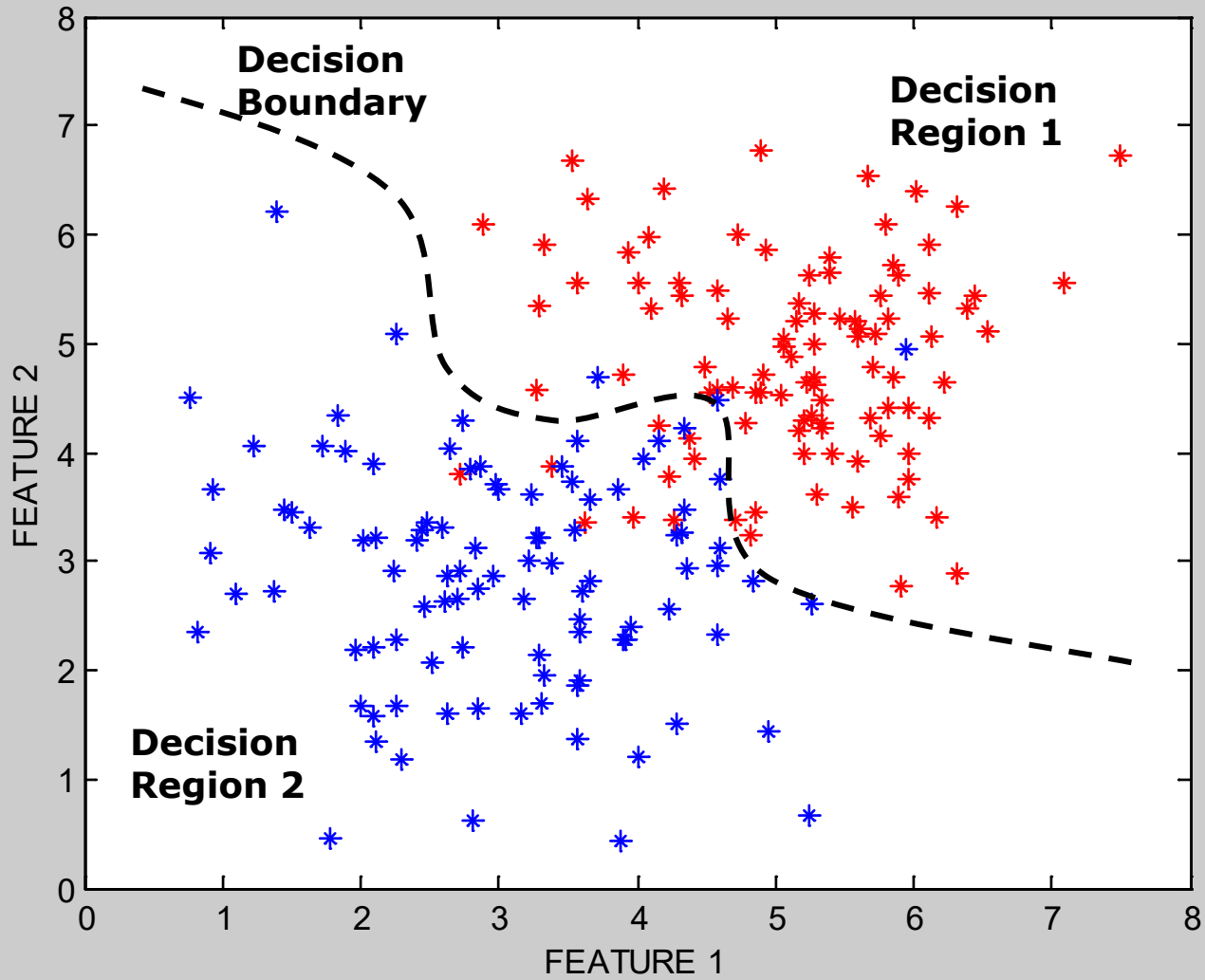
# True Tree (left) versus Learned Tree (right)



# Classification Problem with Overlap



# Decision Boundaries





# Classification in Euclidean Space

- A classifier is a partition of the space  $\underline{x}$  into disjoint decision regions
  - Each region has a label attached
  - Regions with the same label need not be contiguous
  - For a new test point, find what decision region it is in, and predict the corresponding label
- Decision boundaries = boundaries between decision regions
  - The “dual representation” of decision regions
- We can characterize a classifier by the equations for its decision boundaries
- Learning a classifier  $\Leftrightarrow$  searching for the decision boundaries that optimize our objective function

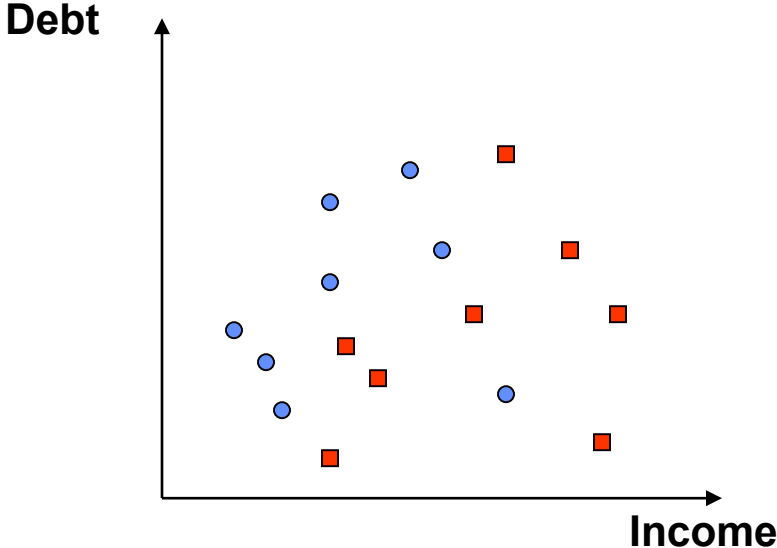
## Example: Decision Trees

- When applied to real-valued attributes, decision trees produce “axis-parallel” linear decision boundaries
- Each internal node is a binary threshold of the form  
 $x_j > t$  ?

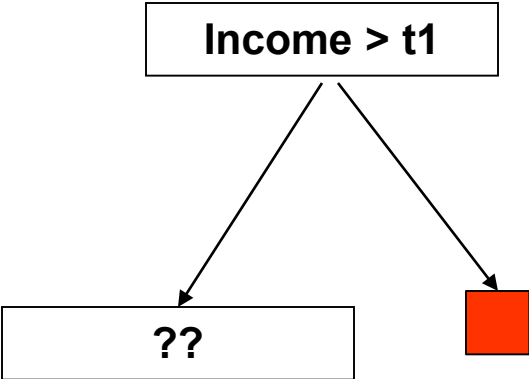
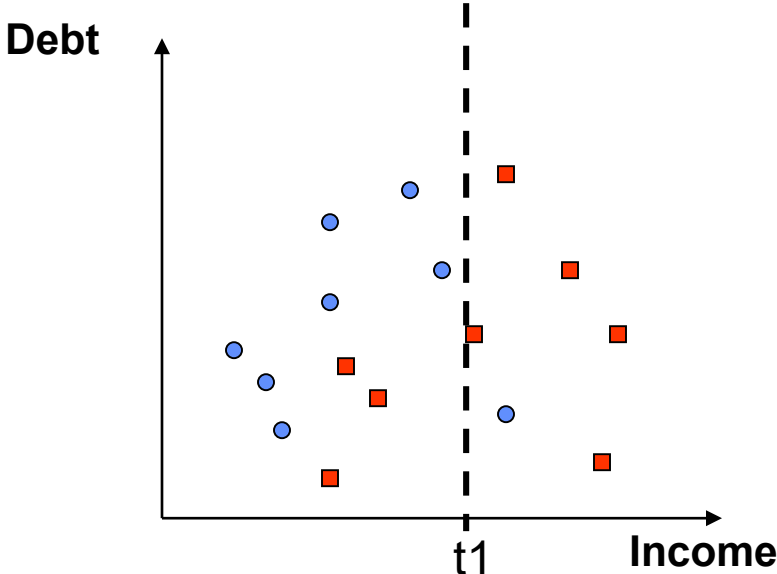
converts each real-valued feature into a binary one

requires evaluation of  $N-1$  possible threshold locations for  $N$  data points, for each real-valued attribute, for each internal node

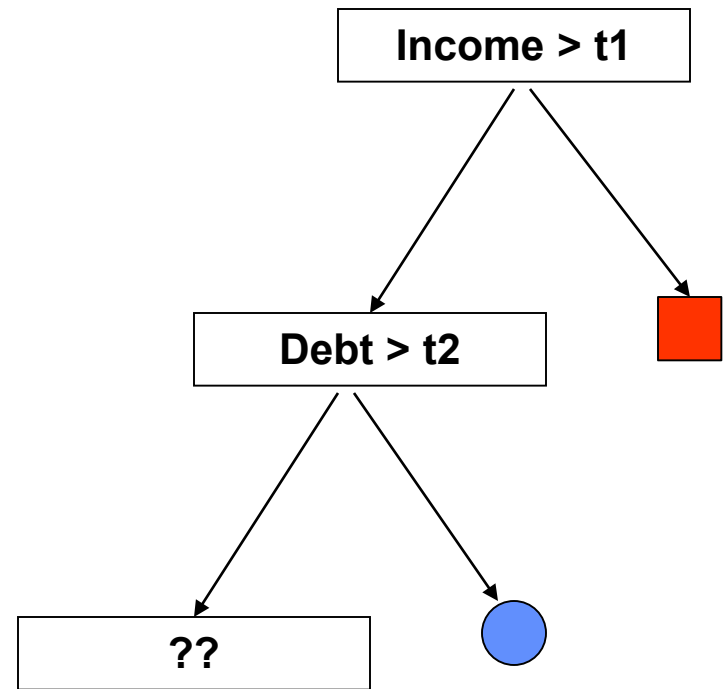
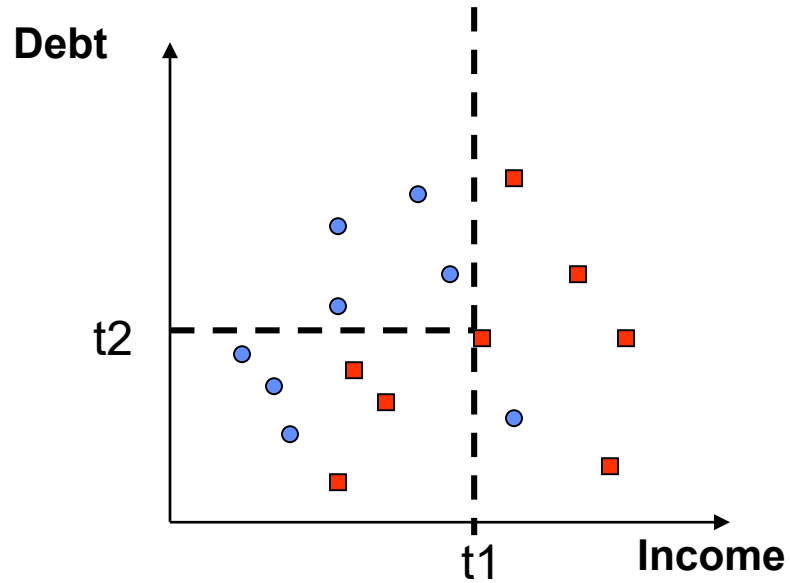
# Decision Tree Example



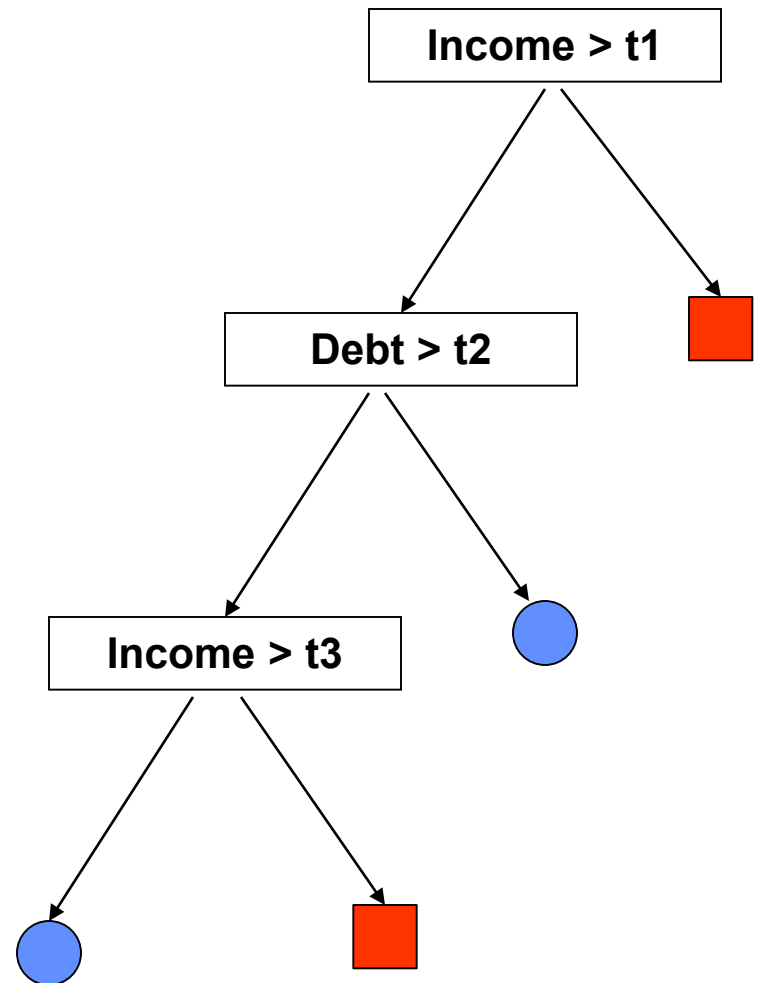
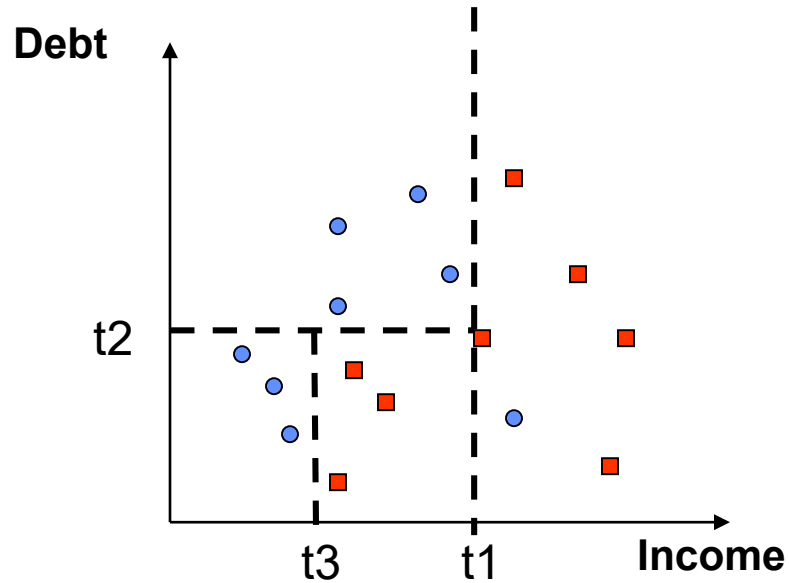
# Decision Tree Example



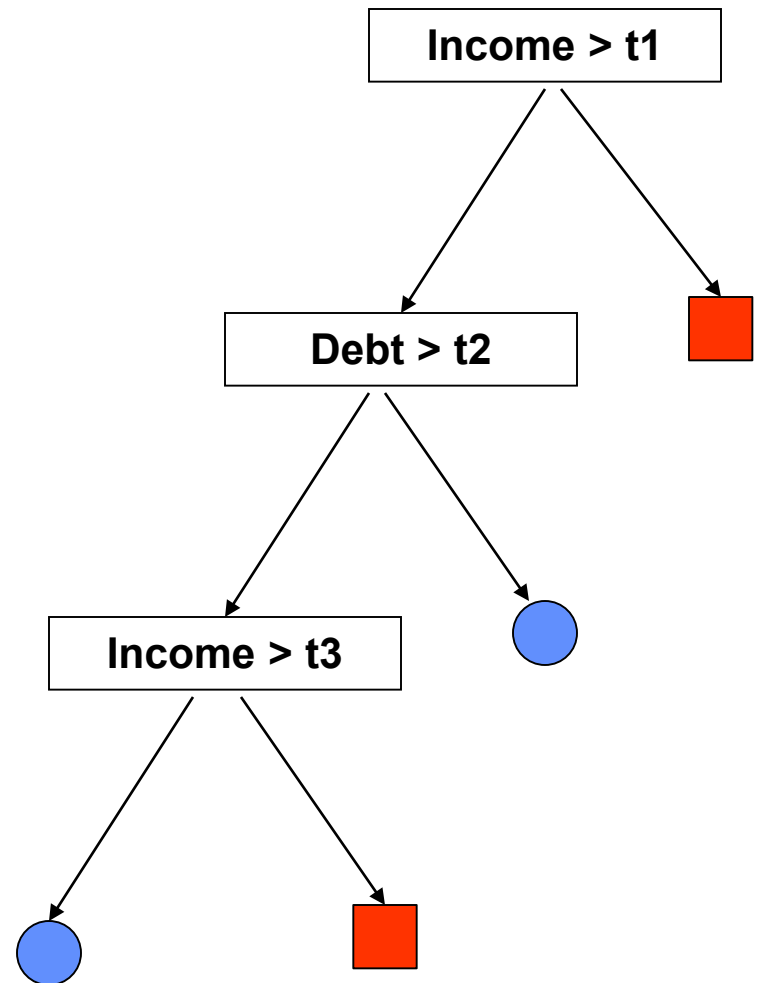
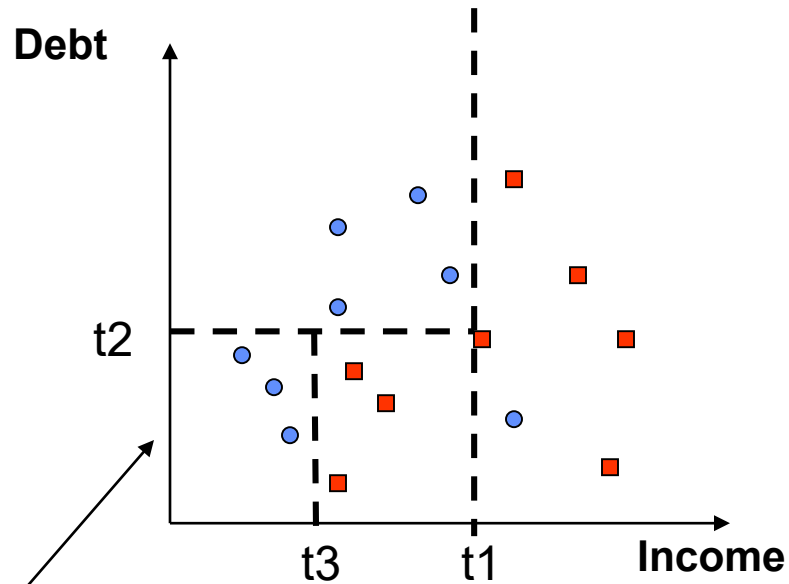
# Decision Tree Example



# Decision Tree Example



# Decision Tree Example



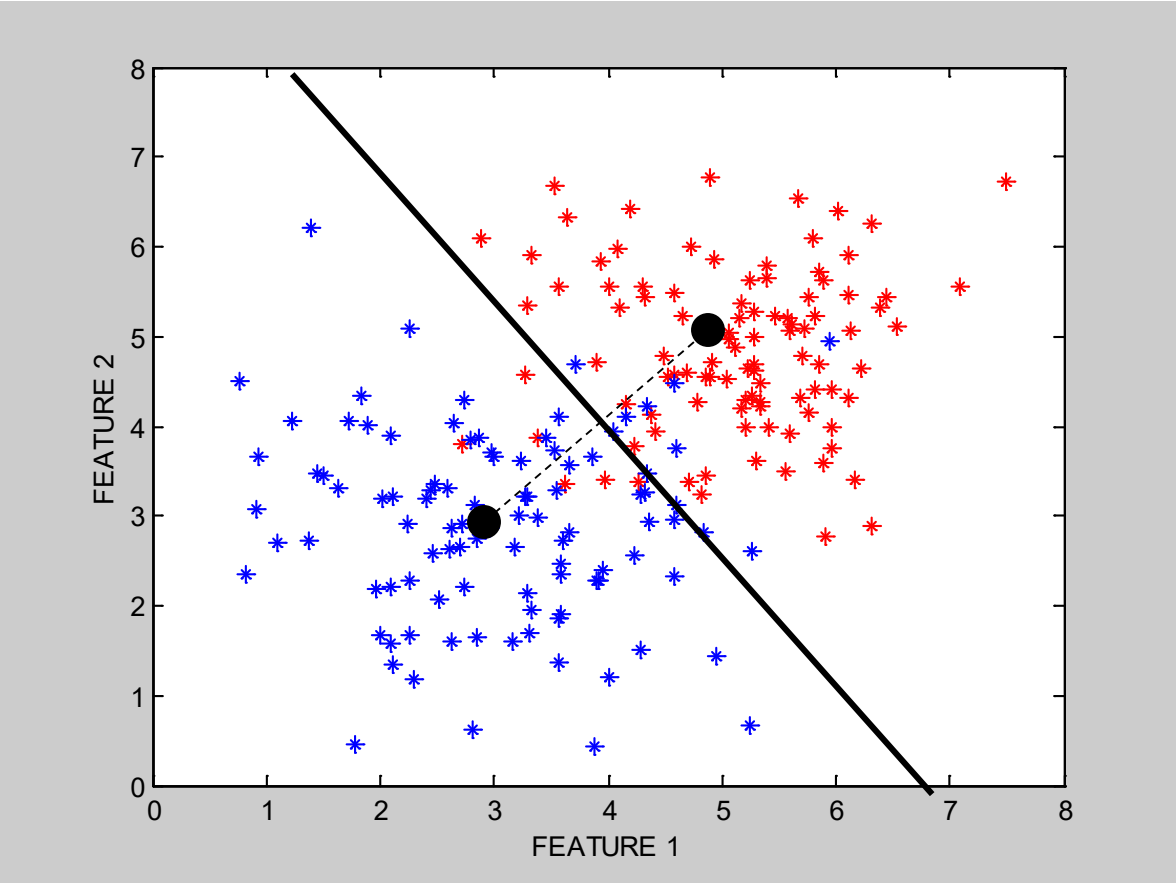
Note: tree boundaries are linear and axis-parallel

## A Simple Classifier: Minimum Distance Classifier

- Training
  - Separate training vectors by class
  - Compute the mean for each class,  $\underline{\mu}_k$ ,  $k = 1, \dots, m$
- Prediction
  - Compute the closest mean to a test vector  $\underline{x}'$  (using Euclidean distance)
  - Predict the corresponding class
- In the 2-class case, the decision boundary is defined by the locus of the hyperplane that is halfway between the 2 means and is orthogonal to the line connecting them
- This is a very simple-minded classifier – easy to think of cases where it will not work very well



# Minimum Distance Classifier



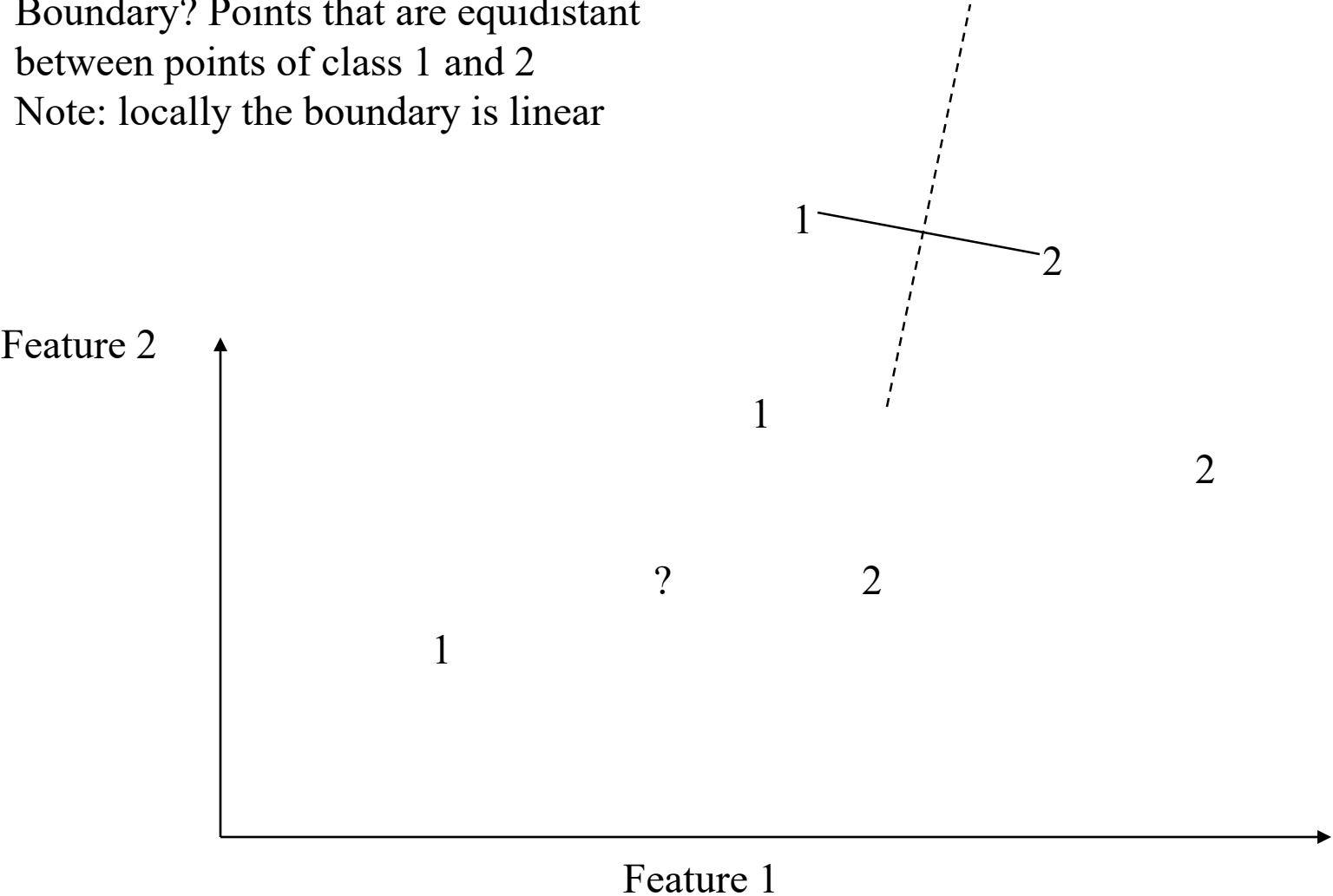
## Another Example: Nearest Neighbor Classifier

- The nearest-neighbor classifier
  - Given a test point  $\underline{x}'$ , compute the distance between  $\underline{x}'$  and each input data point
  - Find the closest neighbor in the training data
  - Assign  $\underline{x}'$  the class label of this neighbor
  - (sort of generalizes minimum distance classifier to exemplars)
- If Euclidean distance is used as the distance measure (the most common choice), the nearest neighbor classifier results in piecewise linear decision boundaries
- Many extensions
  - e.g., kNN, vote based on k-nearest neighbors
  - k can be chosen by cross-validation

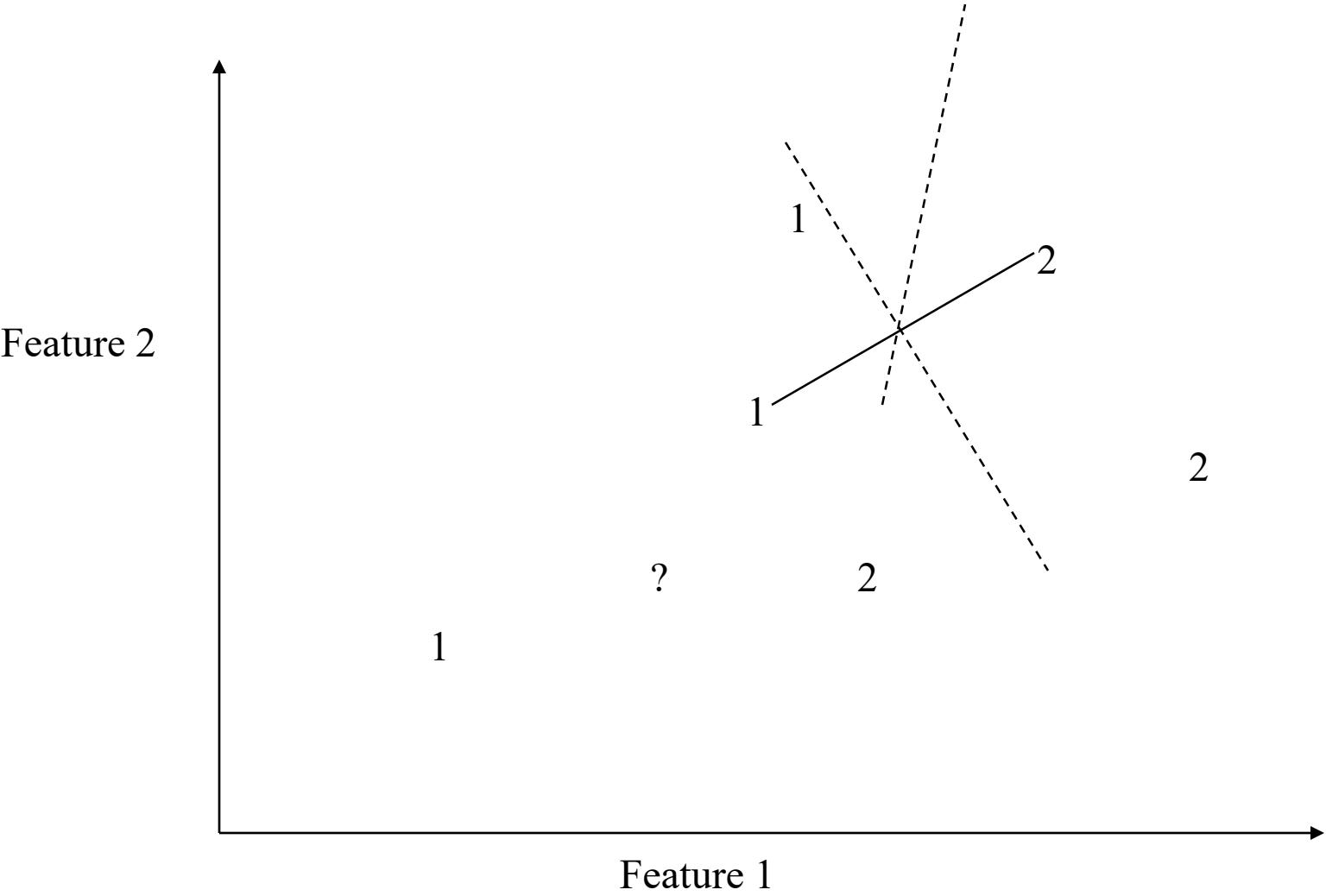
# Local Decision Boundaries

Boundary? Points that are equidistant between points of class 1 and 2

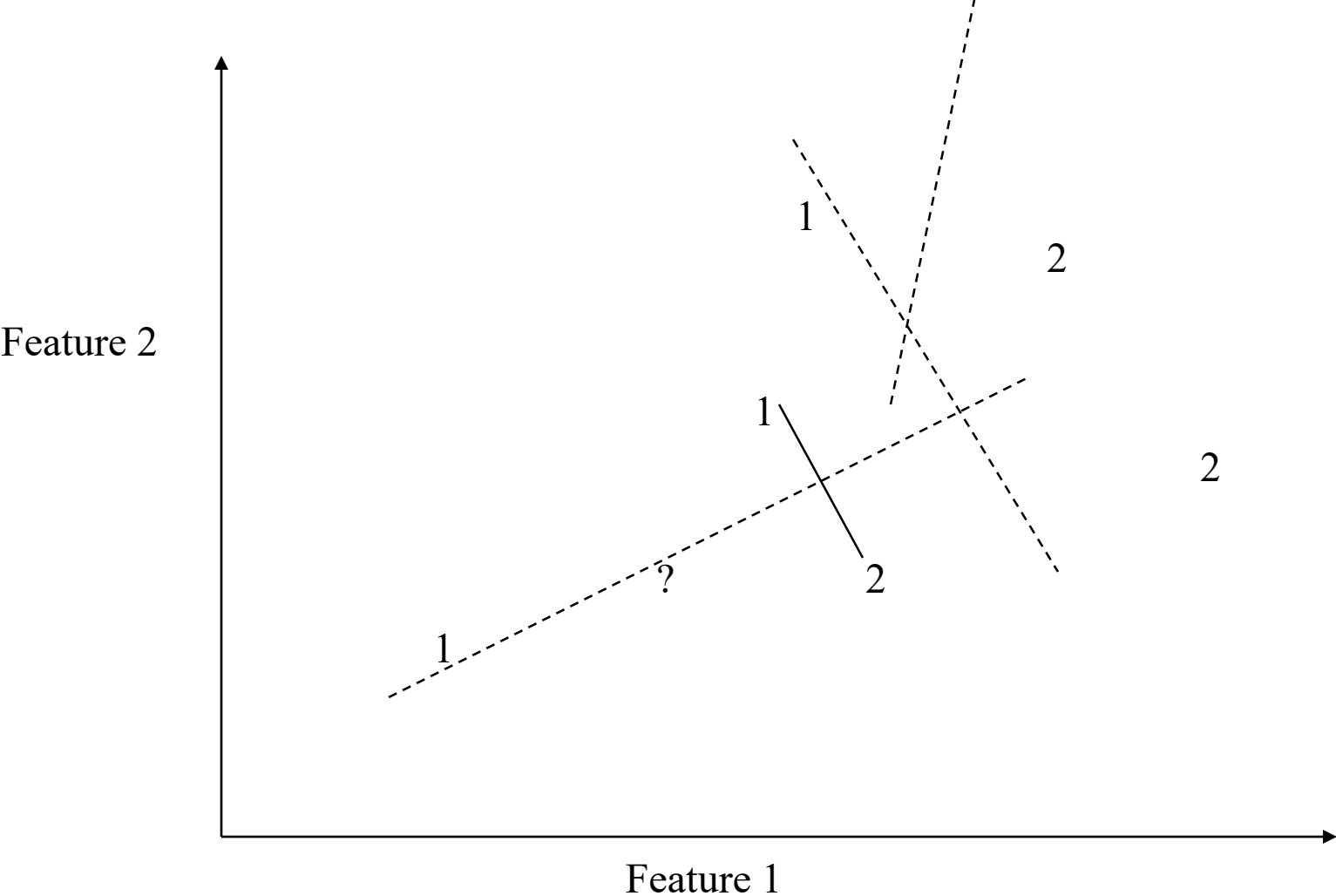
Note: locally the boundary is linear



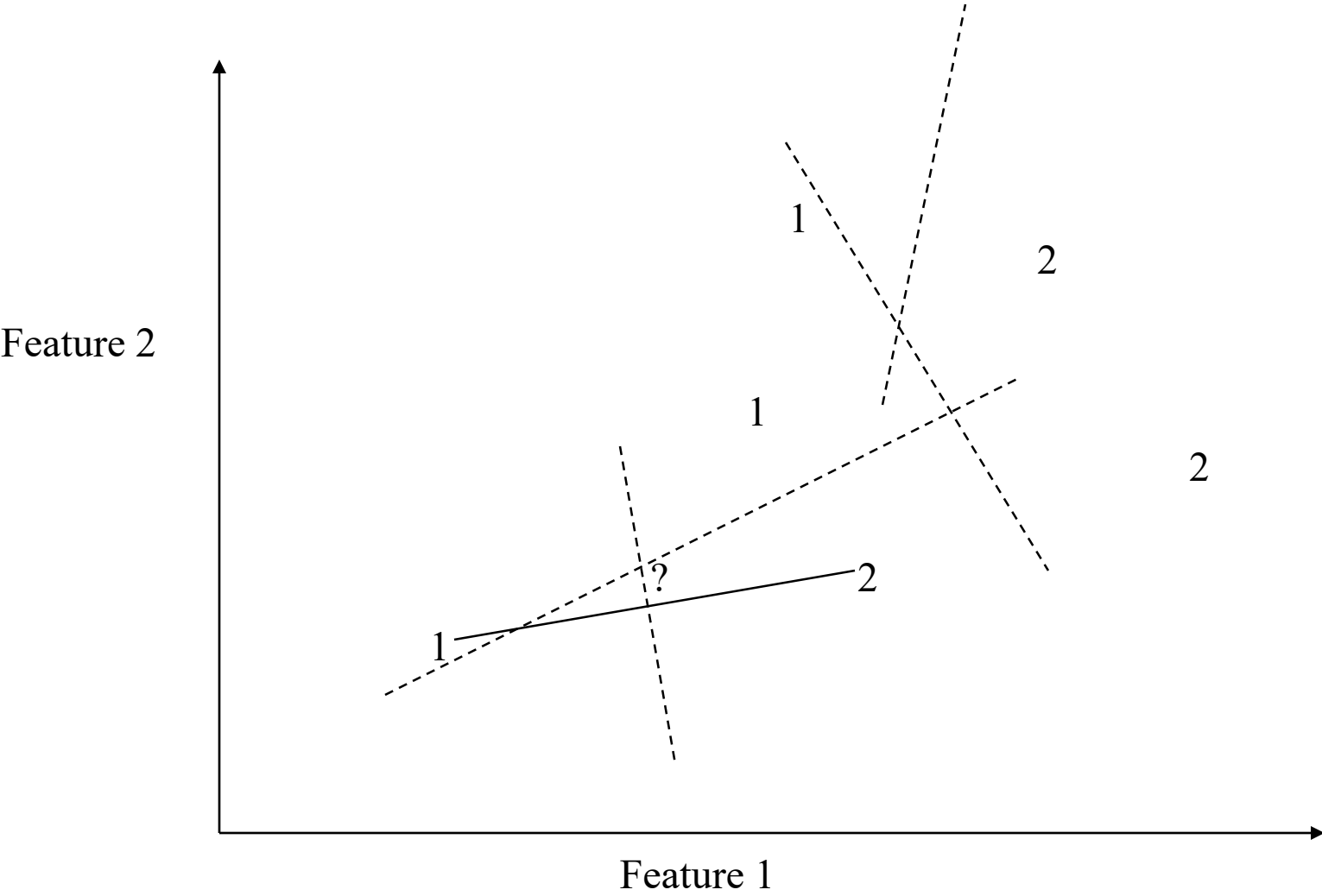
# Finding the Decision Boundaries



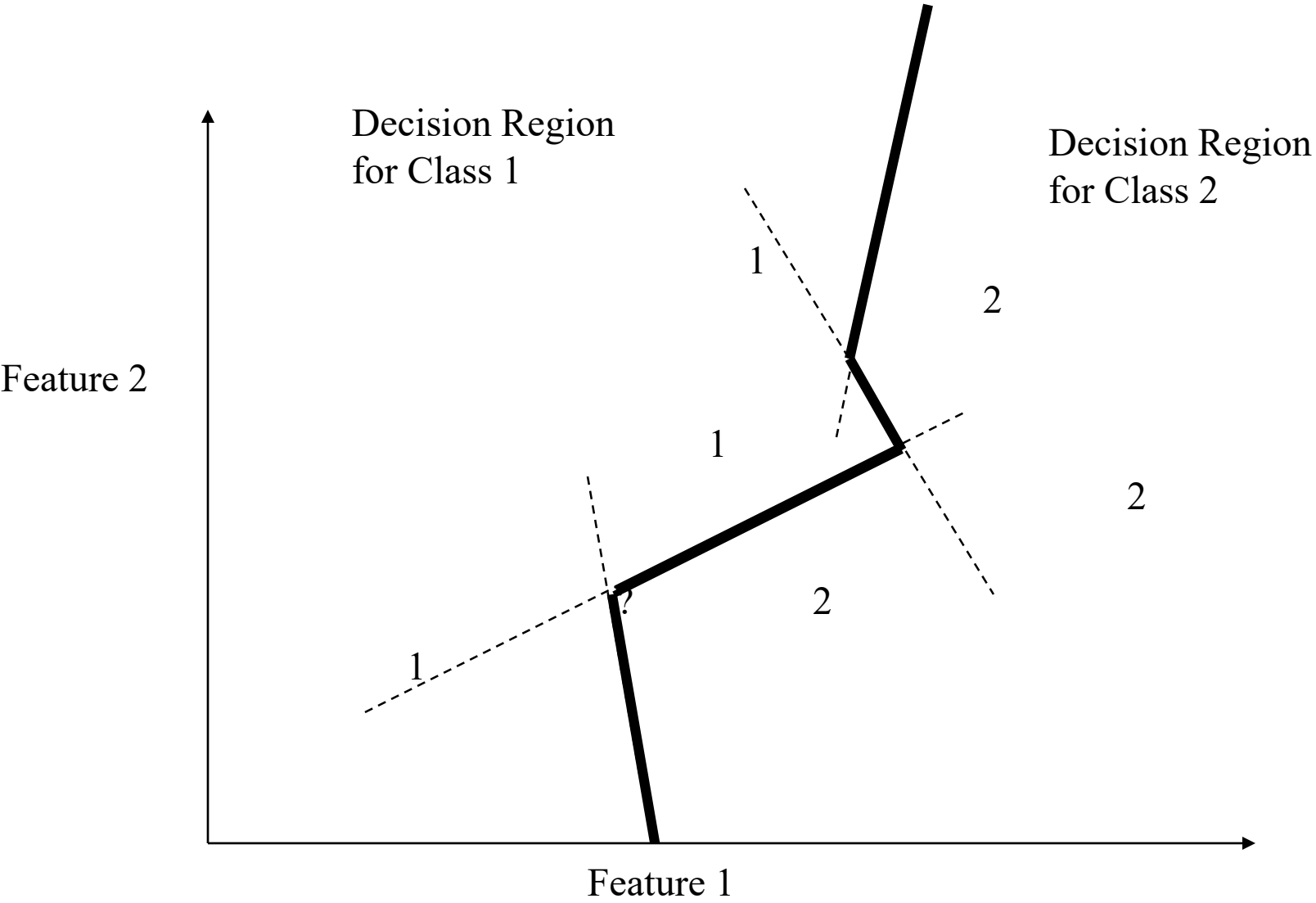
# Finding the Decision Boundaries



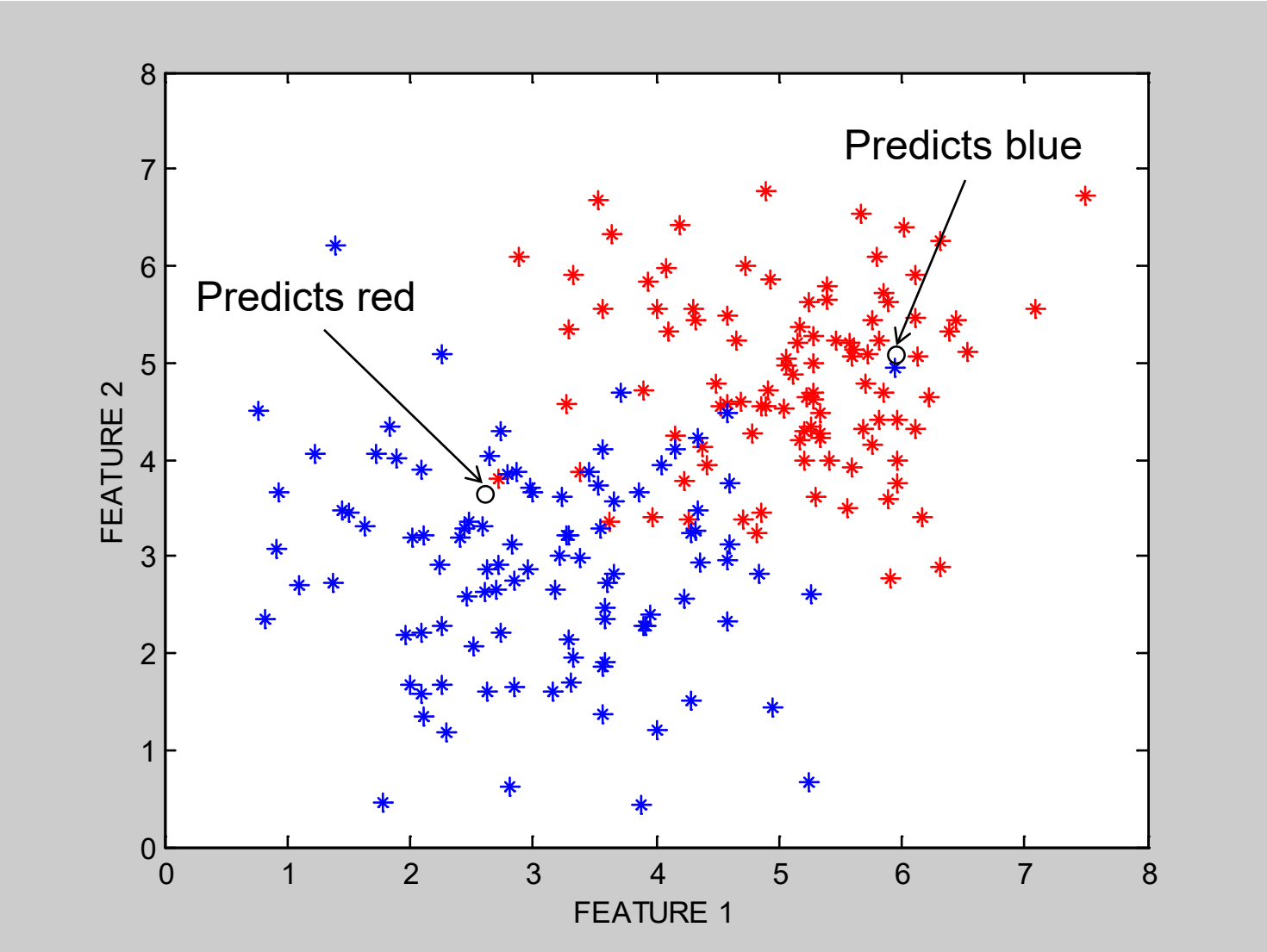
# Finding the Decision Boundaries



# Overall Boundary = Piecewise Linear



# Nearest-Neighbor Boundaries on this data set?

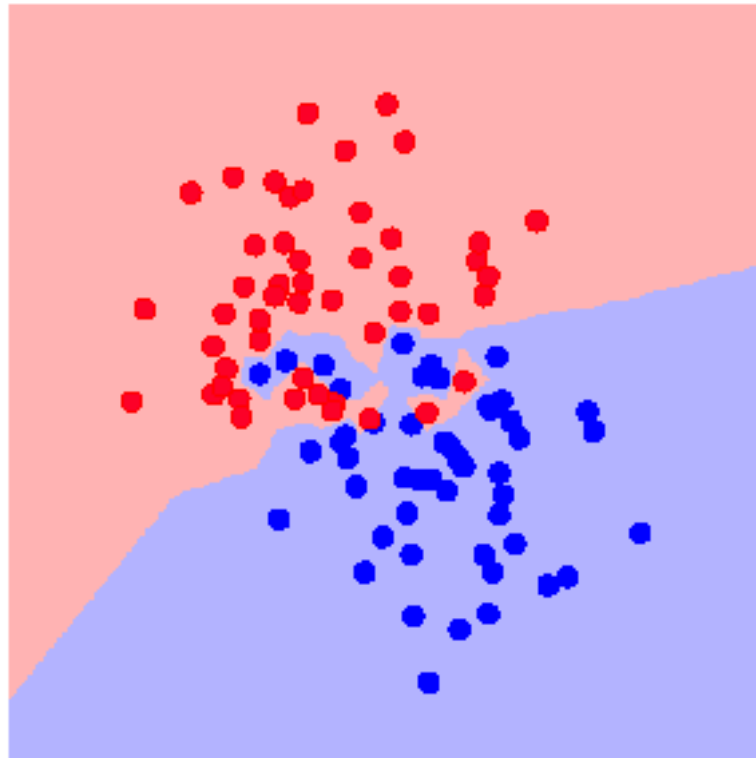




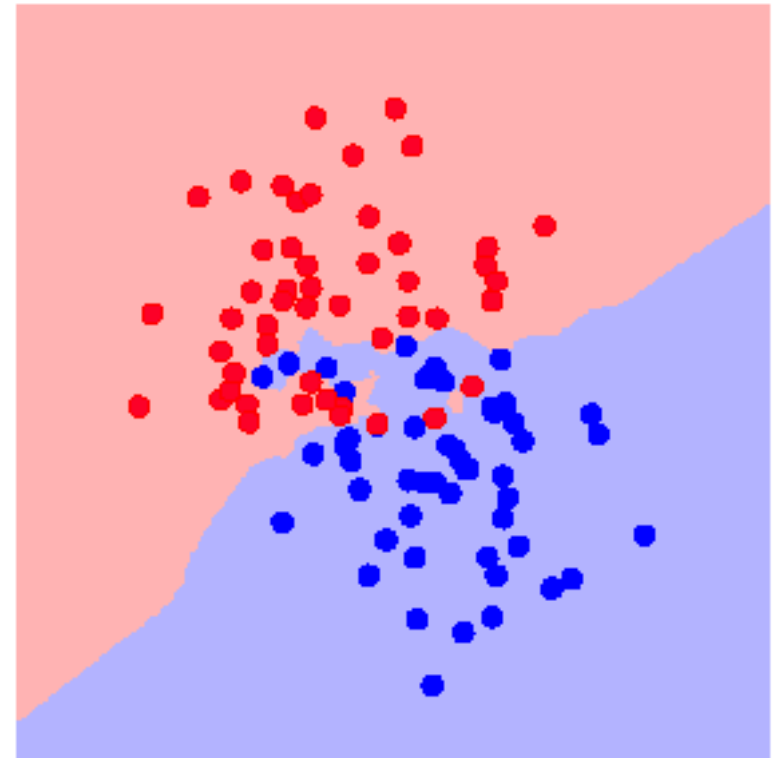
## kNN Decision Boundary

- piecewise linear decision boundary
- Increasing  $k$  "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

$K = 1$



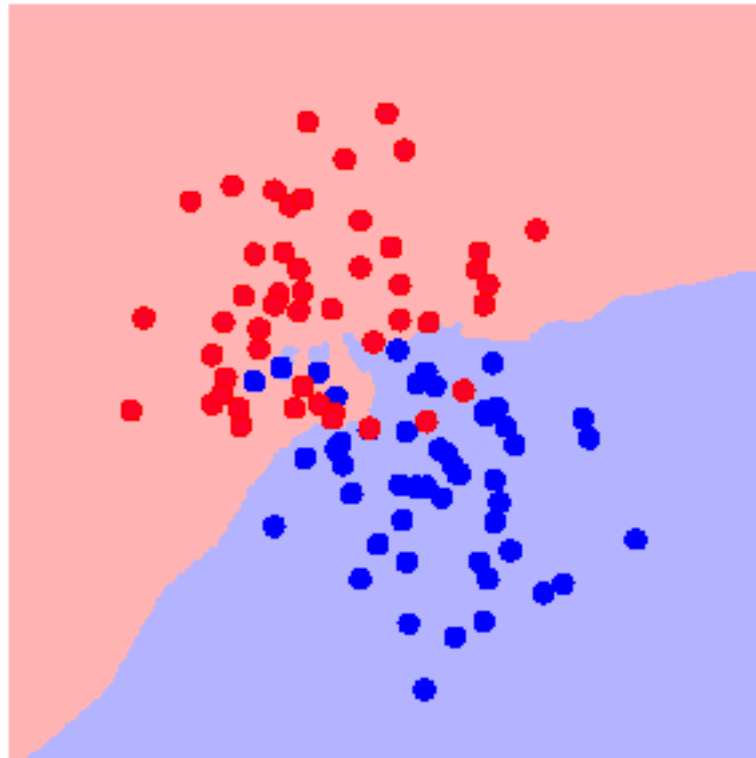
$K = 3$



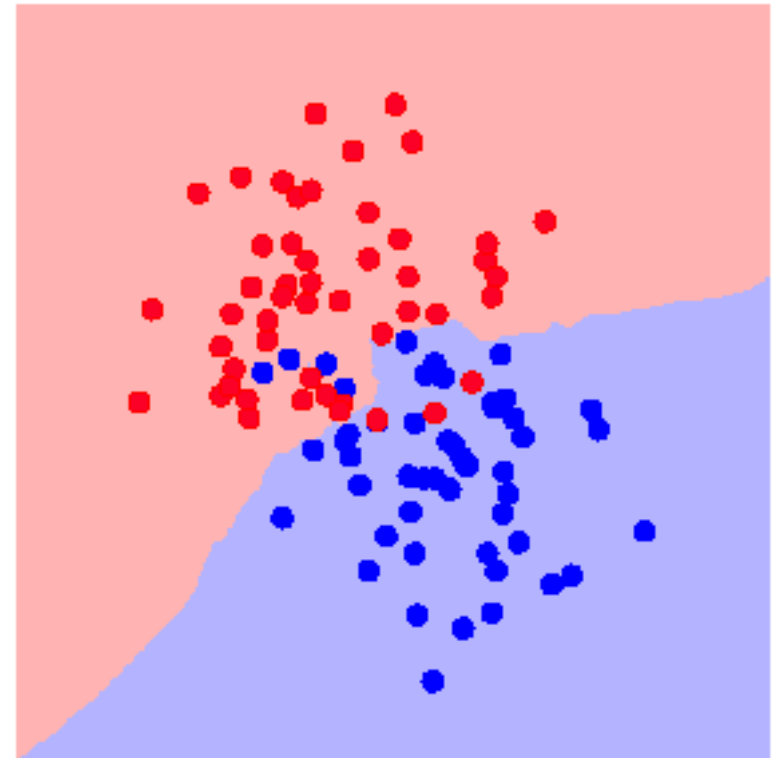
## kNN Decision Boundary

- piecewise linear decision boundary
- Increasing  $k$  "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

$K = 5$



$K = 7$

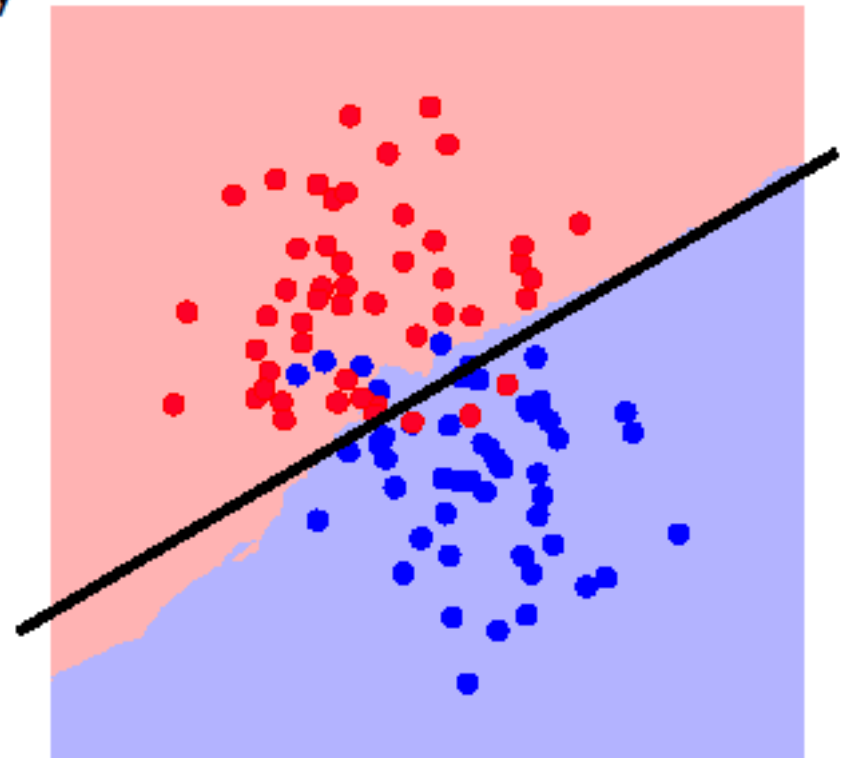


## kNN Decision Boundary

- piecewise linear decision boundary
- Increasing  $k$  "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

- True ("best") decision boundary
  - In this case is linear
  - Compared to kNN: not bad!

$K = 25$

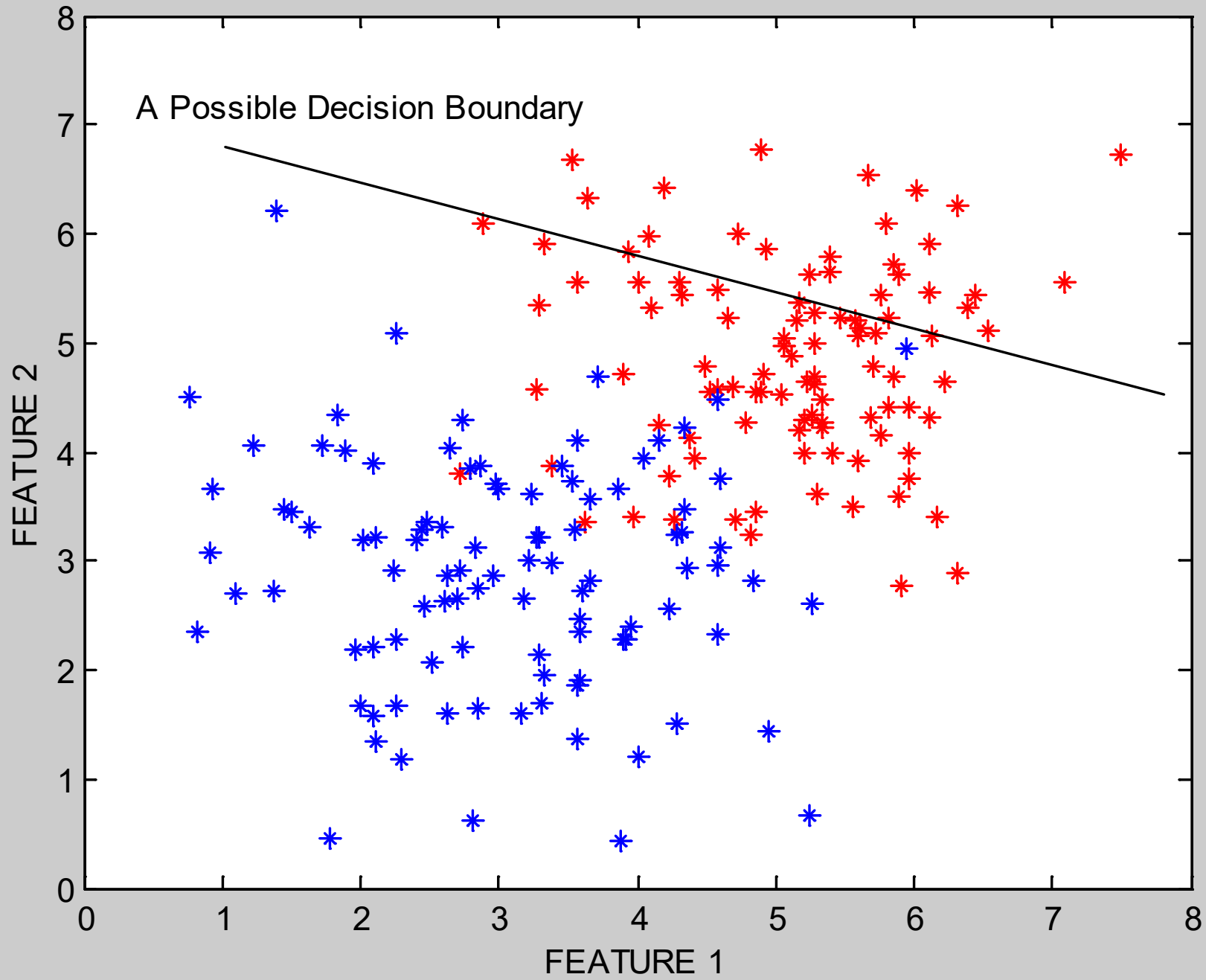


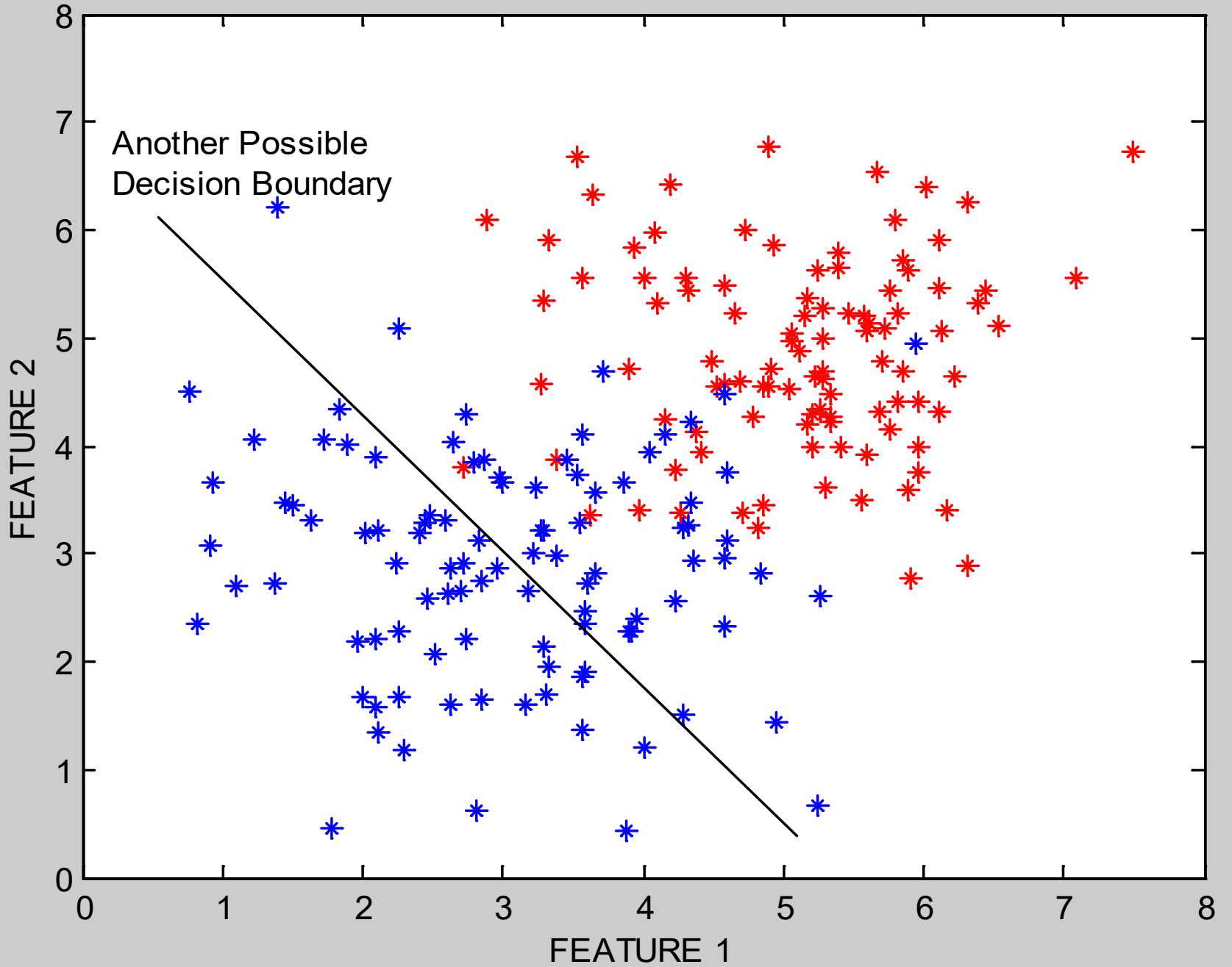
## The kNN Classifier

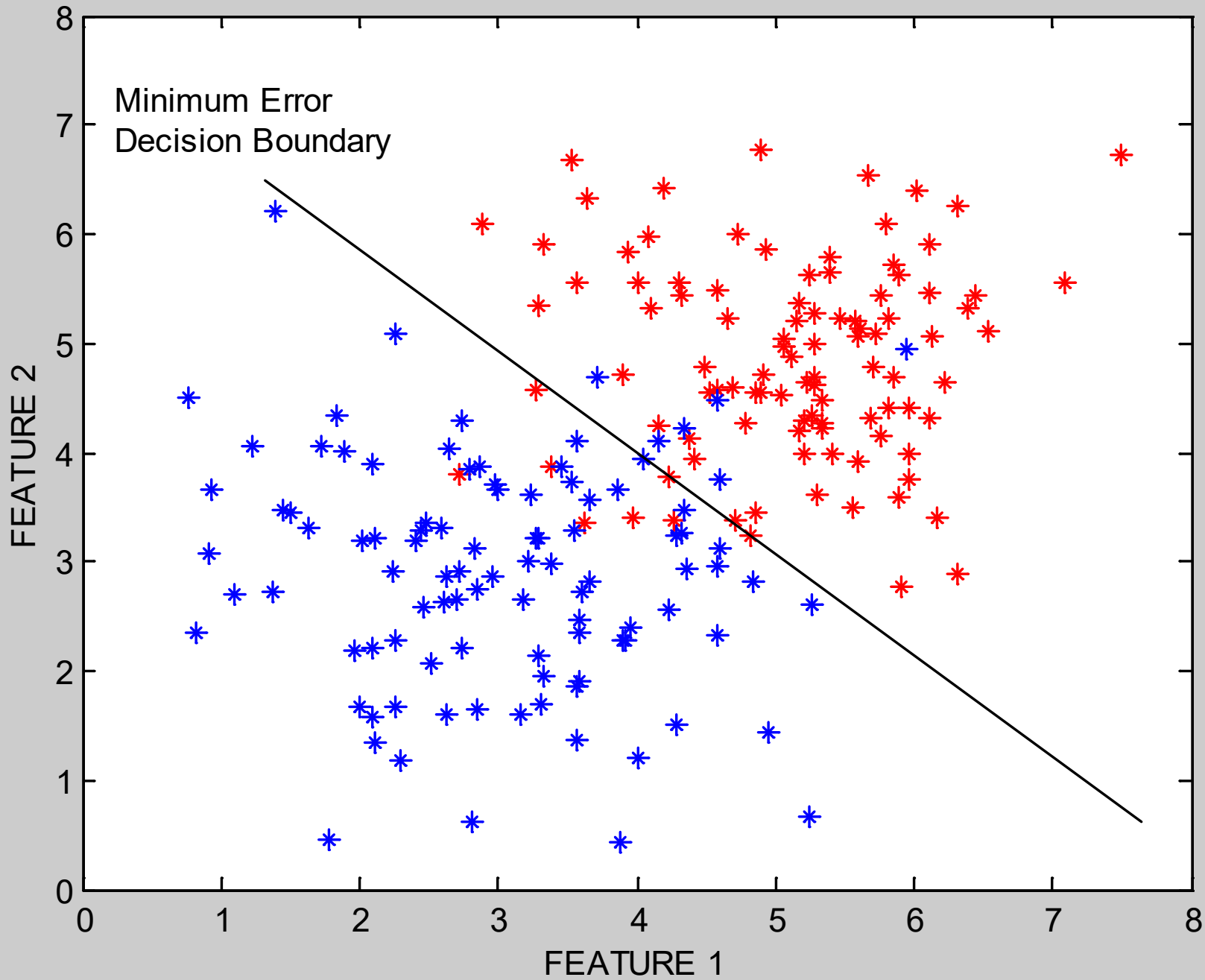
- The kNN classifier often works very well.
- Easy to implement.
- Easy choice if characteristics of your problem are unknown.
- Can be sensitive to the choice of distance metric.
  - Often normalize feature axis values, e.g., z-score or  $[0, 1]$ 
    - E.g., if one feature runs larger in magnitude than another
  - Categorical feature axes are difficult, e.g., Color as Red/Blue/Green
    - Maybe use the absolute differences of their wavelengths?
    - But what about French/Italian/Thai/Burger?
    - Often used:  $\text{delta}(A,B) = \{\text{IF } (A=B) \text{ THEN } 0 \text{ ELSE } 1\}$
- Can encounter problems with sparse training data.
- Can encounter problems in very high dimensional spaces.
  - Most points are corners.
  - Most points are at the edge of the space.
  - Most points are neighbors of most other points.

# Linear Classifiers

- Linear classifier  $\Leftrightarrow$  single linear decision boundary (for 2-class case)
- We can always represent a linear decision boundary by a linear equation:
$$w_1 x_1 + w_2 x_2 + \dots + w_d x_d = \sum w_j x_j = \underline{w}^t \underline{x} = 0$$
- In  $d$  dimensions, this defines a  $(d-1)$  dimensional hyperplane
  - $d=3$ , we get a plane;  $d=2$ , we get a line
- For prediction we simply see if  $\sum w_j x_j > 0$
- The  $w_i$  are the weights (parameters)
  - Learning consists of searching in the  $d$ -dimensional weight space for the set of weights (the linear boundary) that minimizes an error measure
  - A threshold can be introduced by a "dummy" feature that is always one; its weight corresponds to (the negative of) the threshold
- Note that a minimum distance classifier is a special (restricted) case of a linear classifier

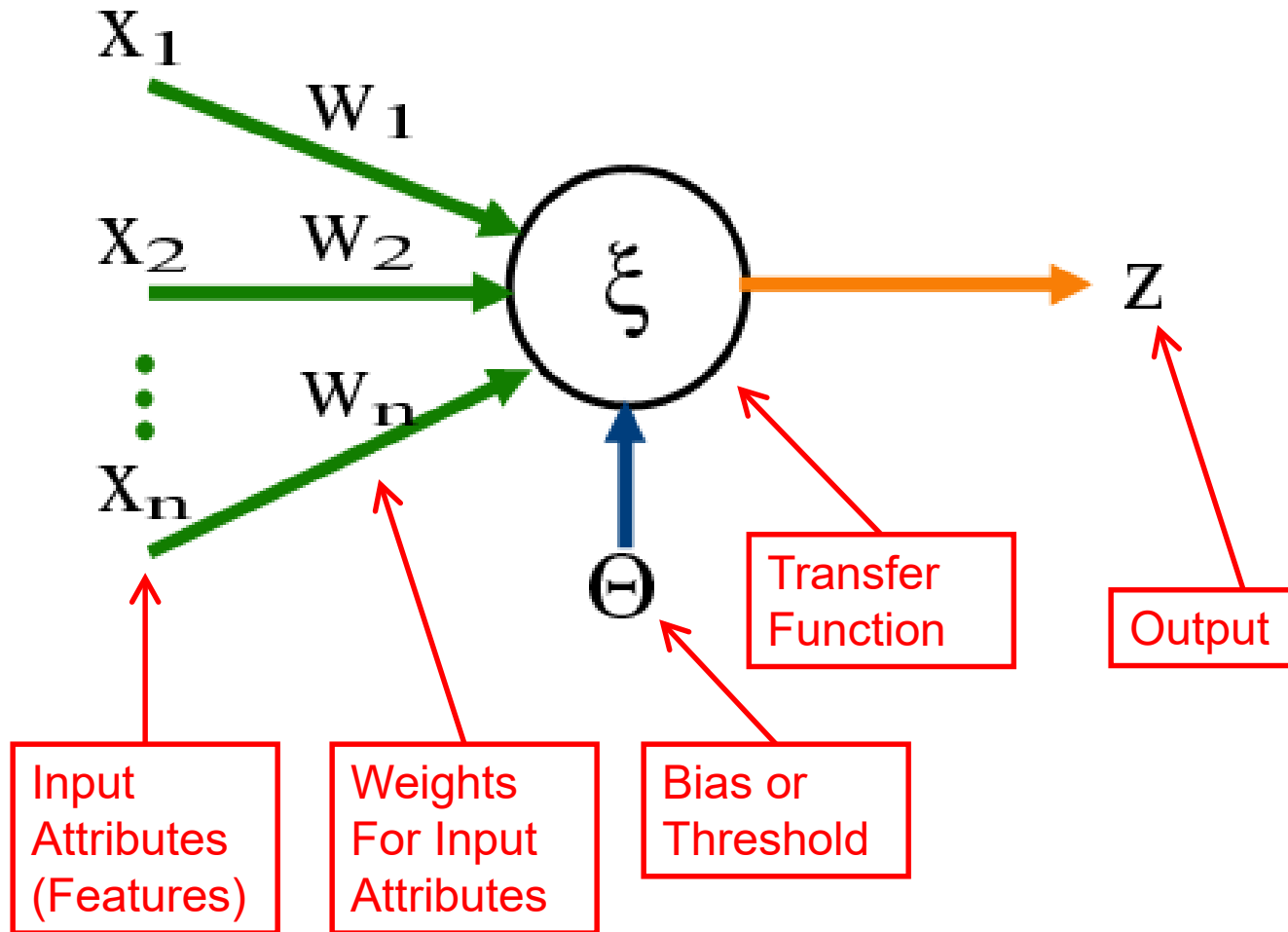








# The Perceptron Classifier (pages 729-731 in text)



## The Perceptron Classifier (pages 729-731 in text)

- The perceptron classifier is just another name for a linear classifier for 2-class data, i.e.,

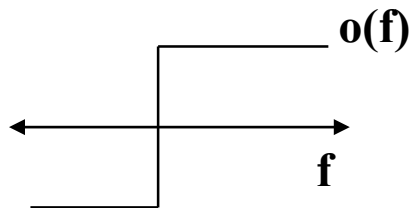
$$\text{output}(\underline{x}) = \text{sign}(\sum w_j x_j)$$

- Loosely motivated by a simple model of how neurons fire
- For mathematical convenience, class labels are +1 for one class and -1 for the other
- Two major types of algorithms for training perceptrons
  - Objective function = classification accuracy (“error correcting”)
  - Objective function = squared error (use gradient descent)
  - Gradient descent is generally faster and more efficient.

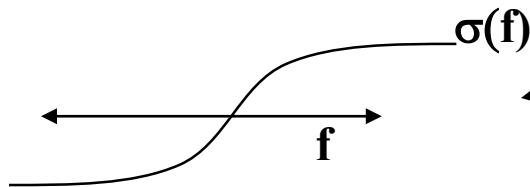
## Two different types of perceptron output

x-axis below is  $f(\underline{x}) = f$  = weighted sum of inputs

y-axis is the perceptron output



Thresholded output (step function),  
takes values +1 or -1



Sigmoid output, takes  
real values between -1 and +1

The sigmoid is in effect an approximation  
to the threshold function above, but  
has a gradient that we can use for learning

- Sigmoid function is defined as

$$\sigma[f] = [ 2 / ( 1 + \exp[- f ] ) ] - 1$$

- Derivative of sigmoid

$$\partial\sigma/\partial f [ f ] = .5 * ( \sigma[f]+1 ) * ( 1-\sigma[f] )$$

## Squared Error for Perceptron with Sigmoidal Output

- Squared error =  $E[\underline{w}] = \sum_i [ \sigma(f[\underline{x}(i)]) - y(i) ]^2$

where  $\underline{x}(i)$  is the  $i$ th input vector in the training data,  $i=1,\dots,N$   
 $y(i)$  is the  $i$ th target value (-1 or 1)

$f[\underline{x}(i)] = \sum w_j x_j$  is the weighted sum of inputs  
 $\sigma(f[\underline{x}(i)])$  is the sigmoid of the weighted sum

- Note that everything is fixed (once we have the training data) except for the weights  $\underline{w}$
- So we want to minimize  $E[\underline{w}]$  as a function of  $\underline{w}$

# Gradient Descent Learning of Weights

Gradient Descent Rule:

$$\underline{\mathbf{w}}_{\text{new}} = \underline{\mathbf{w}}_{\text{old}} - \eta \Delta ( \mathbf{E}[\underline{\mathbf{w}}] )$$

where

$\Delta ( \mathbf{E}[\underline{\mathbf{w}}] )$  is the gradient of the error function  $E$  wrt weights, and  $\eta$  is the learning rate (small, positive)

Notes:

1. This moves us downhill in direction  $\Delta ( \mathbf{E}[\underline{\mathbf{w}}] )$  (steepest downhill)
2. How far we go is determined by the value of  $\eta$

## Gradient Descent Update Equation

- From basic calculus, for perceptron with sigmoid, and squared error objective function, gradient for a single input  $\underline{x}(i)$  is

$$\Delta ( E[\underline{w}] ) = - ( y(i) - \sigma[f(i)] ) \partial\sigma[f(i)] x_j(i)$$

- Gradient descent weight update rule:

$$w_j = w_j + \eta ( y(i) - \sigma[f(i)] ) \partial\sigma[f(i)] x_j(i)$$

- can rewrite as:

$$w_j = w_j + \eta * \text{error} * c * x_j(i)$$

## Pseudo-code for Perceptron Training

```
Initialize each  $w_j$  (e.g., randomly)
```

```
While (termination condition not satisfied)
```

```
  for  $i = 1 : N$  % loop over data points (an iteration)
```

```
    for  $j = 1 : d$  % loop over weights
```

```
       $\text{deltaw}_j = \eta ( y(i) - \sigma[f(i)] ) \partial\sigma[f(i)] x_j(i)$ 
```

```
       $w_j = w_j + \text{deltaw}_j$ 
```

```
    end
```

```
  calculate termination condition
```

```
end
```

- Inputs:  $N$  features,  $N$  targets (class labels), learning rate  $\eta$
- Outputs: a set of learned weights

# Comments on Perceptron Learning

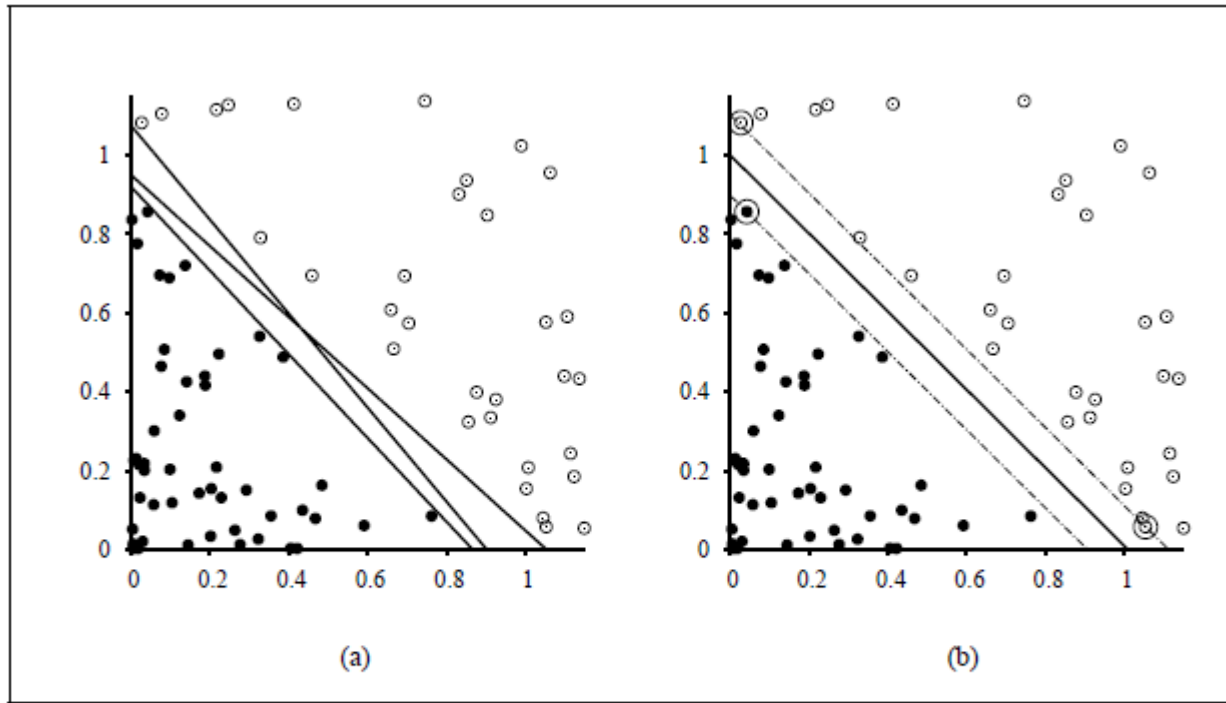
- Iteration = one pass through all of the data
- Algorithm presented = incremental gradient descent
  - Weights are updated after visiting each input example
  - Alternatives
    - Batch: update weights after each iteration (typically slower)
    - Stochastic: randomly select examples and then do weight updates
- A similar iterative algorithm learns weights for thresholded output (step function) perceptrons
- Rate of convergence
  - $E[\underline{w}]$  is convex as a function of  $\underline{w}$ , so no local minima
  - So convergence is guaranteed as long as learning rate is small enough
    - But if we make it too small, learning will be *very* slow
  - But if learning rate is too large, we move further, but can overshoot the solution and oscillate, and not converge at all



# Support Vector Machines (SVM): “Modern perceptrons” (section 18.9, R&N)

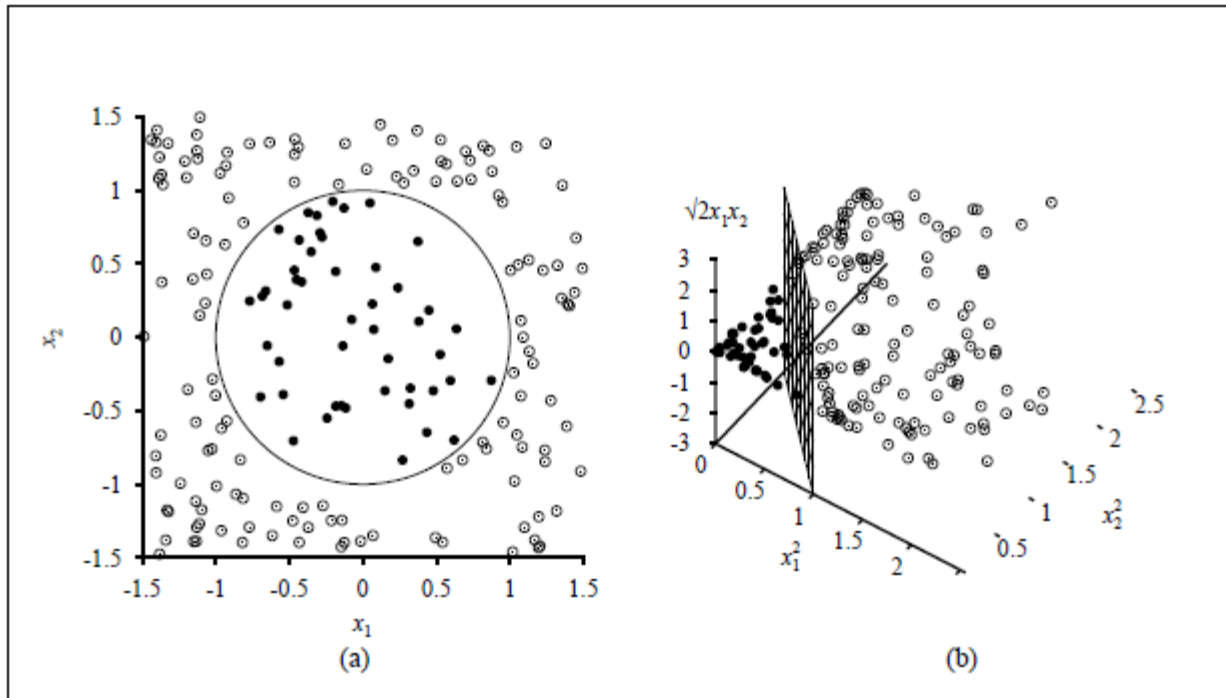
- A modern linear separator classifier
  - Essentially, a perceptron with a few extra wrinkles
- Constructs a **“maximum margin separator”**
  - A linear decision boundary with the largest possible distance from the decision boundary to the example points it separates
  - “Margin” = Distance from decision boundary to closest example
  - The “maximum margin” helps SVMs to generalize well
- Can embed the data in a non-linear higher dimension space
  - Constructs a linear separating hyperplane in that space
    - **This can be a non-linear boundary in the original space**
  - Algorithmic advantages and simplicity of linear classifiers
  - Representational advantages of non-linear decision boundaries
- **Currently most popular “off-the shelf” supervised classifier.**

## Constructs a "maximum margin separator"



**Figure 18.30** FILES: . Support vector machine classification: (a) Two classes of points (black and white circles) and three candidate linear separators. (b) The maximum margin separator (heavy line), is at the midpoint of the **margin** (area between dashed lines). The **support vectors** (points with large circles) are the examples closest to the separator.

# Can embed the data in a non-linear higher dimension space



**Figure 18.31** FILES: . (a) A two-dimensional training set with positive examples as black circles and negative examples as white circles. The true decision boundary,  $x_1^2 + x_2^2 \leq 1$ , is also shown. (b) The same data after mapping into a three-dimensional input space  $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$ . The circular decision boundary in (a) becomes a linear decision boundary in three dimensions. Figure 18.29(b) gives a closeup of the separator in (b).

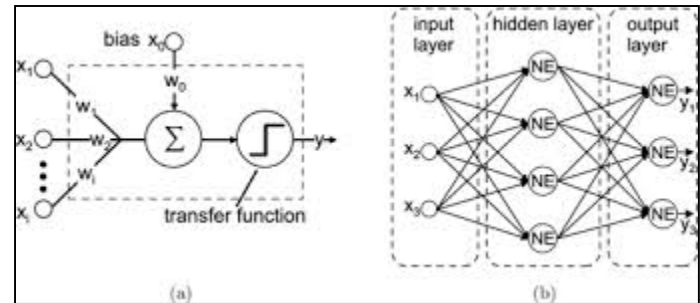
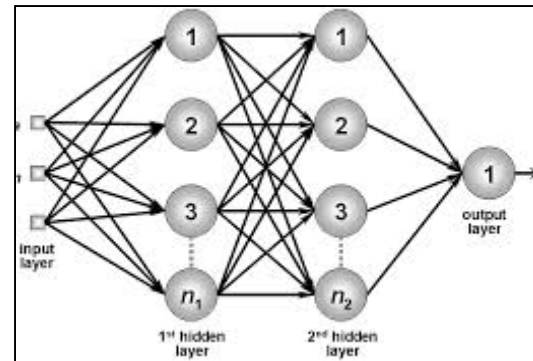
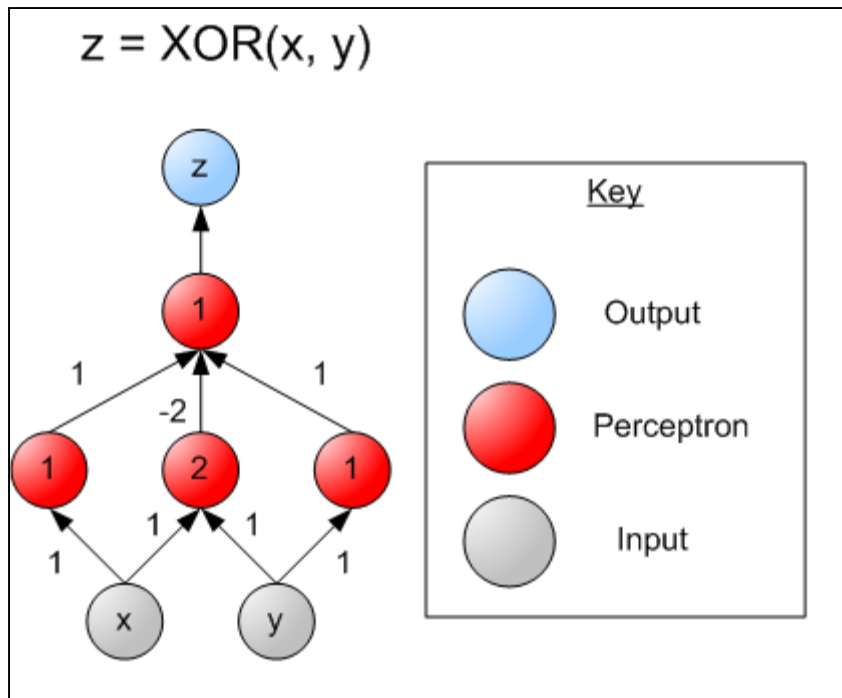
# Multi-Layer Perceptrons (Artificial Neural Networks)

(sections 18.7.3-18.7.4 in textbook)

- What if we took  $K$  perceptrons and trained them in parallel and then took a weighted sum of their sigmoidal outputs?
  - This is a multi-layer neural network with a single “hidden” layer (the outputs of the first set of perceptrons)
  - If we train them jointly in parallel, then intuitively different perceptrons could learn different parts of the solution
    - They define different local decision boundaries in the input space
- What if we hooked them up into a general Directed Acyclic Graph?
  - Can create simple “neural circuits” (but no feedback; not fully general)
  - Often called neural networks with hidden units
- How would we train such a model?
  - Backpropagation algorithm = clever way to do gradient descent
  - Bad news: many local minima and many parameters
    - training is hard and slow
  - Good news: can learn general non-linear decision boundaries
  - Generated much excitement in AI in the late 1980’s and 1990’s
  - New current excitement with very large “deep learning” networks

# Multi-Layer Perceptrons (Artificial Neural Networks)

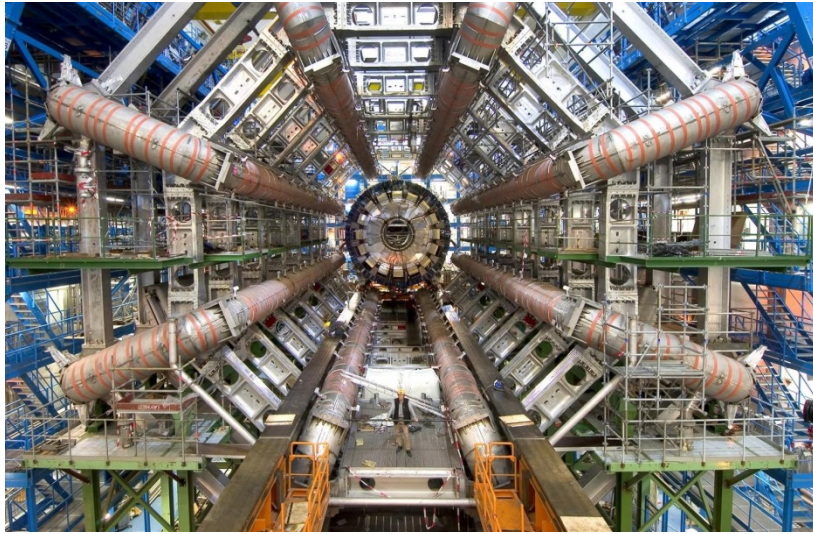
(sections 18.7.3-18.7.4 in textbook)



Thanks to  
Pierre Baldi

# Cutting Edge of Machine Learning: Deep Learning in Neural Networks

Thanks to  
Xiaohui Xie



F

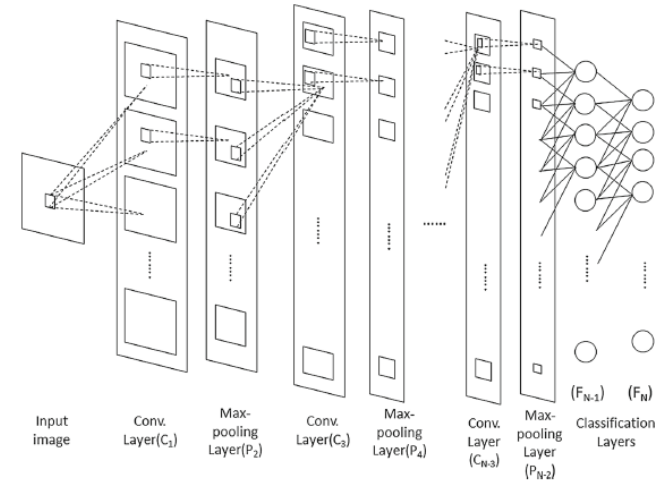
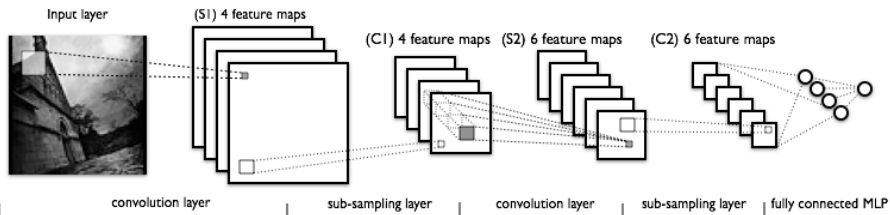


Fig. 2 The overall architecture of the DCNN used by us, which includes an input layer, multiple alternating convolution and max-pooling layers, and two fully connected classification layers.  $N$  denotes the total number of layers in the network



PRO:

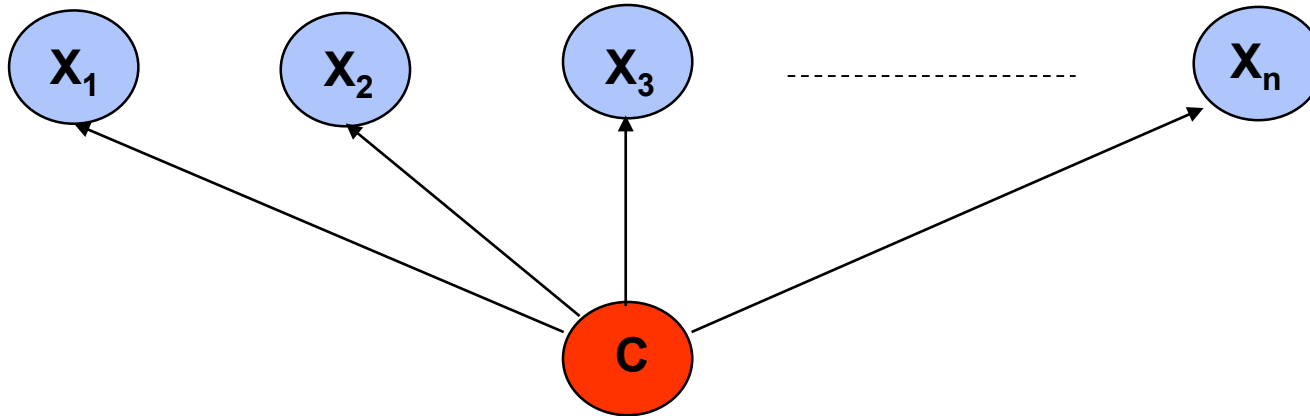
- \* Good results on hard problems
- \* Combine feature extraction with classification directly from image

CON:

- \* Can be difficult to train; gradient descent does not work well
- \* Can be slow to train; but fast computers and modern techniques help

# Naïve Bayes Model

(section 20.2.2 R&N 3<sup>rd</sup> ed.)



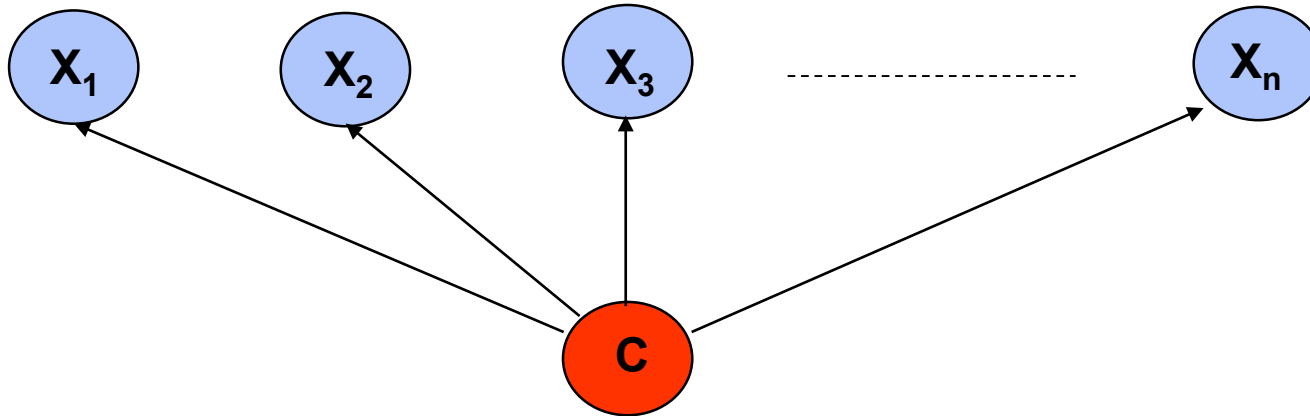
**Basic Idea:** We want to estimate  $P(C | X_1, \dots, X_n)$ , but it's hard to think about computing the probability of a class from input attributes of an example.

**Solution:** Use Bayes' Rule to turn  $P(C | X_1, \dots, X_n)$  into a proportionally equivalent expression that involves only  $P(C)$  and  $P(X_1, \dots, X_n | C)$ . Then assume that feature values are conditionally independent given class, which allows us to turn  $P(X_1, \dots, X_n | C)$  into  $\prod_i P(X_i | C)$ .

We estimate  $P(C)$  easily from the frequency with which each class appears within our training data, and we estimate  $P(X_i | C)$  easily from the frequency with which each  $X_i$  appears in each class  $C$  within our training data.

# Naïve Bayes Model

(section 20.2.2 R&N 3<sup>rd</sup> ed.)



**Bayes Rule:**  $P(C | X_1, \dots, X_n)$  is proportional to  $P(C) \prod_i P(X_i | C)$   
[note: denominator  $P(X_1, \dots, X_n)$  is constant for all classes, may be ignored.]

Features  $X_i$  are conditionally independent given the class variable  $C$

- choose the class value  $c_i$  with the highest  $P(c_i | x_1, \dots, x_n)$
- simple to implement, often works very well
- e.g., spam email classification:  $X$ 's = counts of words in emails

Conditional probabilities  $P(X_i | C)$  can easily be estimated from labeled data

- Problem: Need to avoid zeroes, e.g., from limited training data
- Solutions: Pseudo-counts, beta[a,b] distribution, etc.



## Naïve Bayes Model (2)

$$P(C | X_1, \dots, X_n) = \alpha P(C) \prod_i P(X_i | C)$$

Probabilities  $P(C)$  and  $P(X_i | C)$  can easily be estimated from labeled data

$$P(C = c_j) \approx \#(\text{Examples with class label } C = c_j) / \#(\text{Examples})$$

$$P(X_i = x_{ik} | C = c_j) \\ \approx \#(\text{Examples with attribute value } X_i = x_{ik} \text{ and class label } C = c_j) \\ / \#(\text{Examples with class label } C = c_j)$$

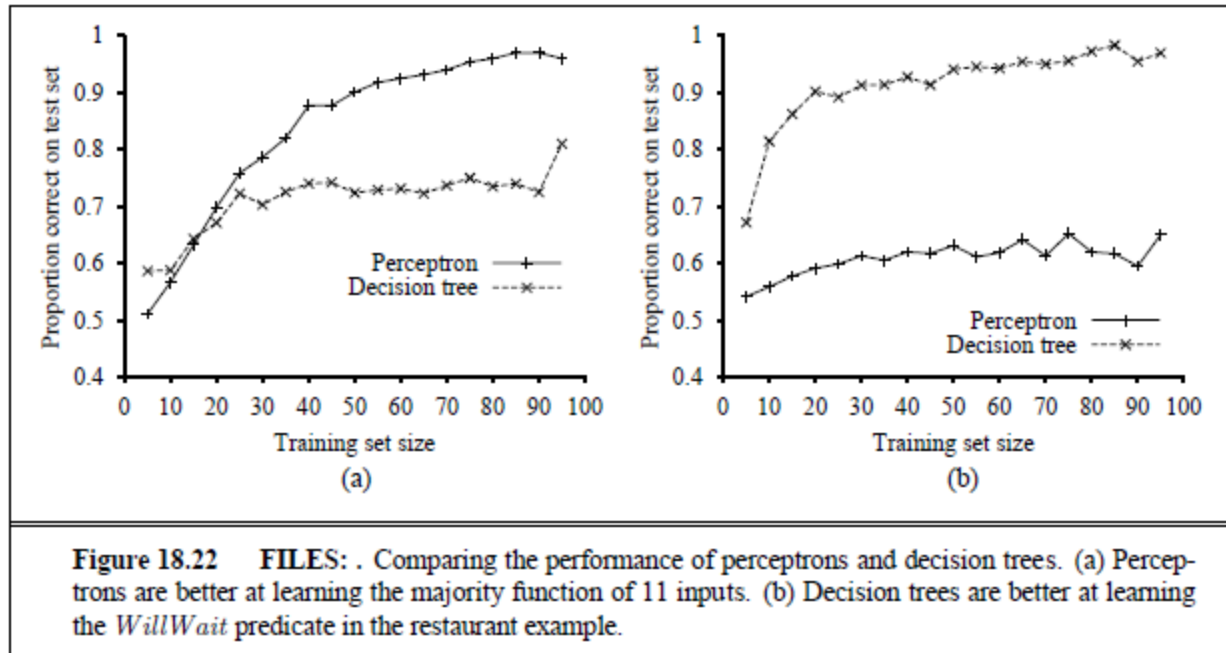
Usually easiest to work with logs

$$\log [ P(C | X_1, \dots, X_n) ] \\ = \log \alpha + \log P(C) + \sum \log P(X_i | C)$$

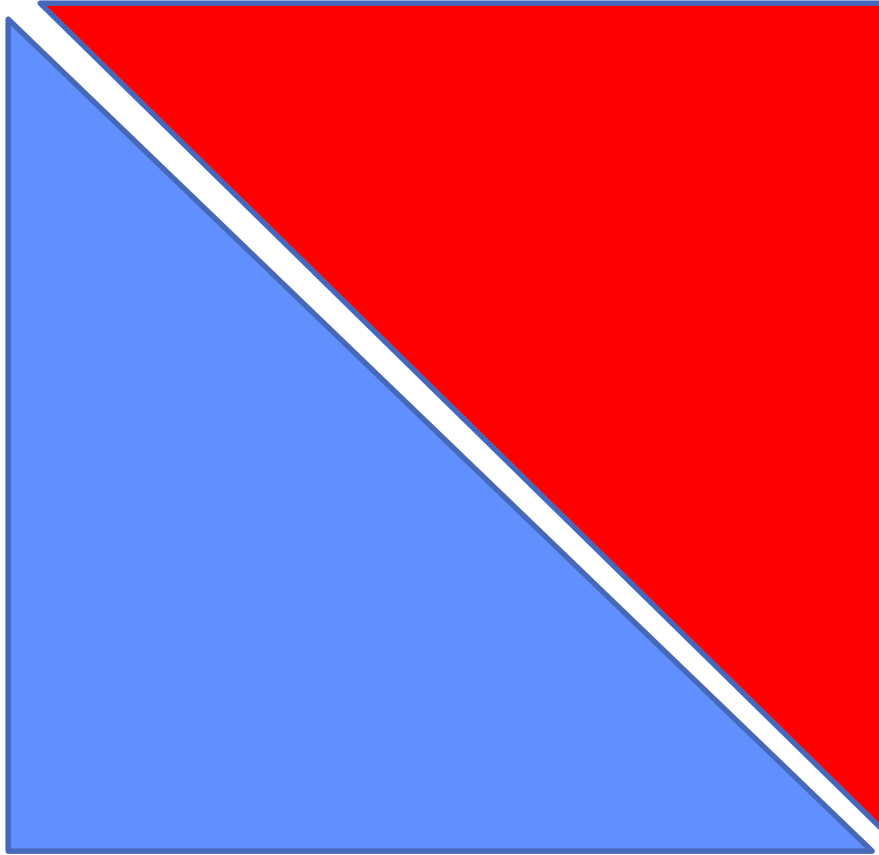
**DANGER:** What if ZERO examples with value  $X_i = x_{ik}$  and class label  $C = c_j$  ?  
An unseen example with value  $X_i = x_{ik}$  will NEVER predict class label  $C = c_j$  !

Practical solutions: Pseudocounts, e.g., add 1 to every  $\#()$  , etc.  
Theoretical solutions: Bayesian inference, beta distribution, etc.

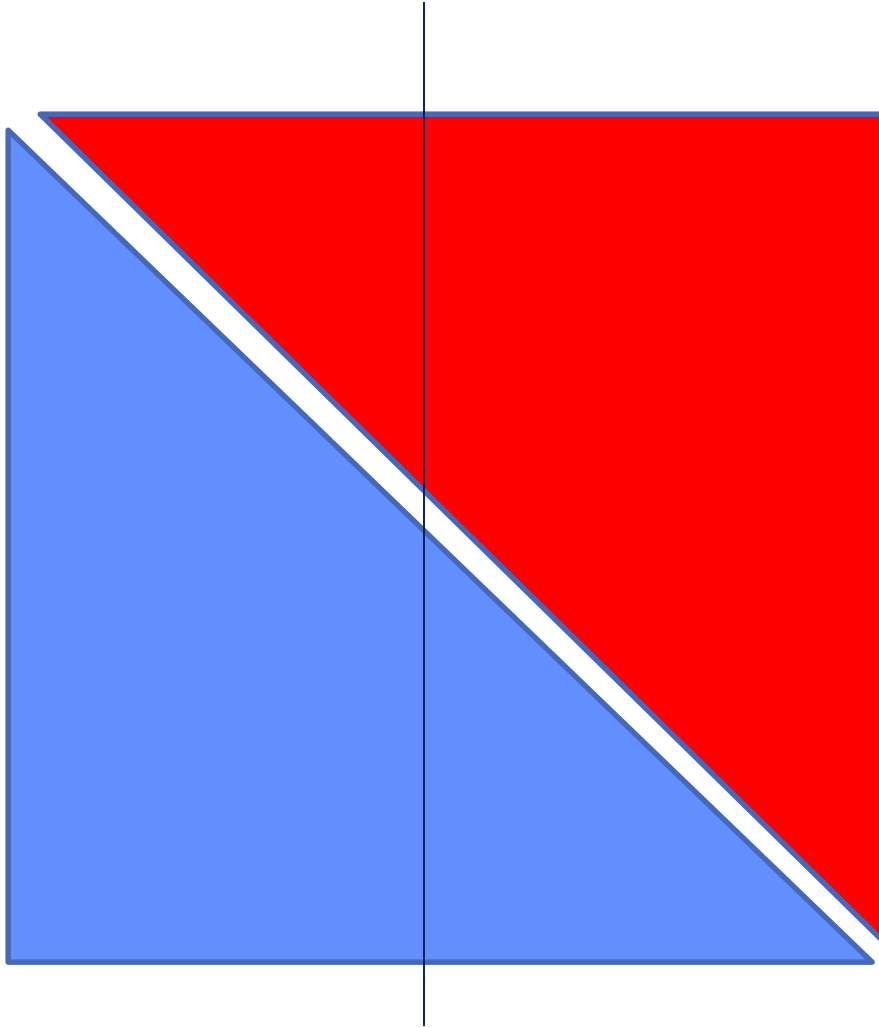
# Classifier Bias – Decision Tree or Linear Perceptron?



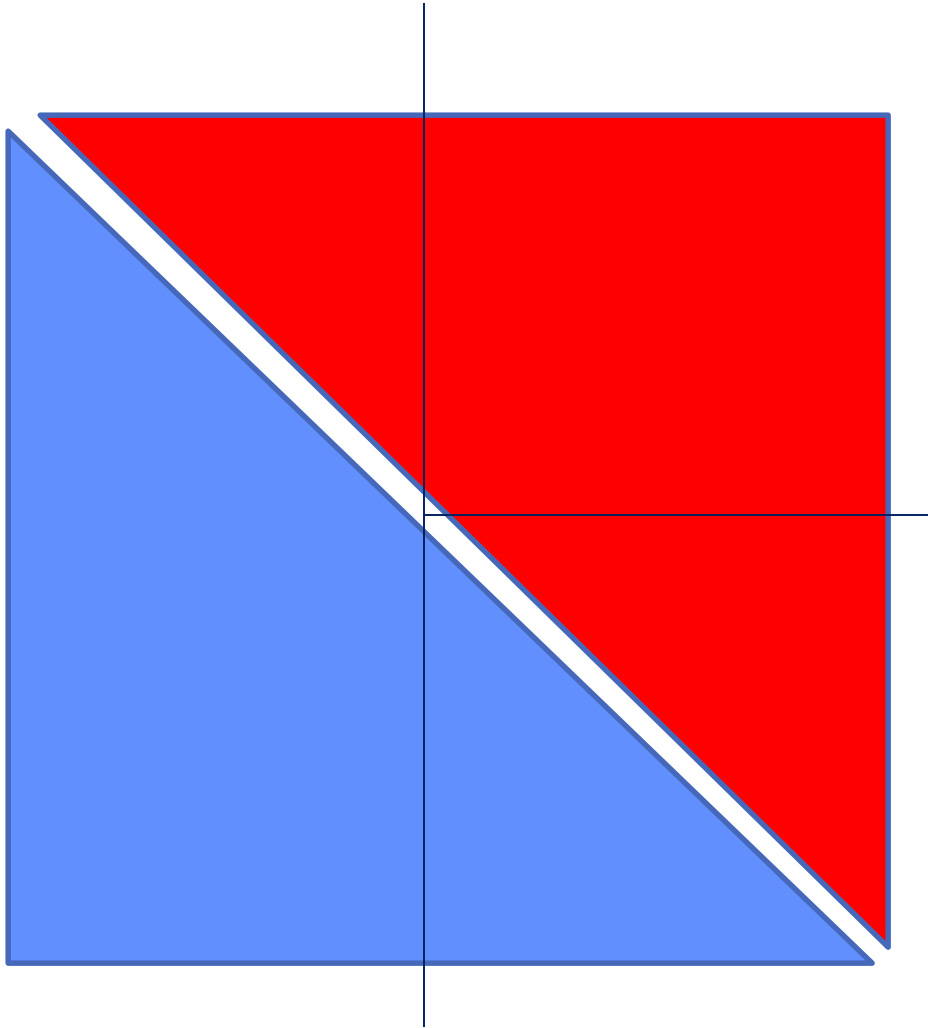
## Classifier Bias – Decision Tree or Linear Perceptron?



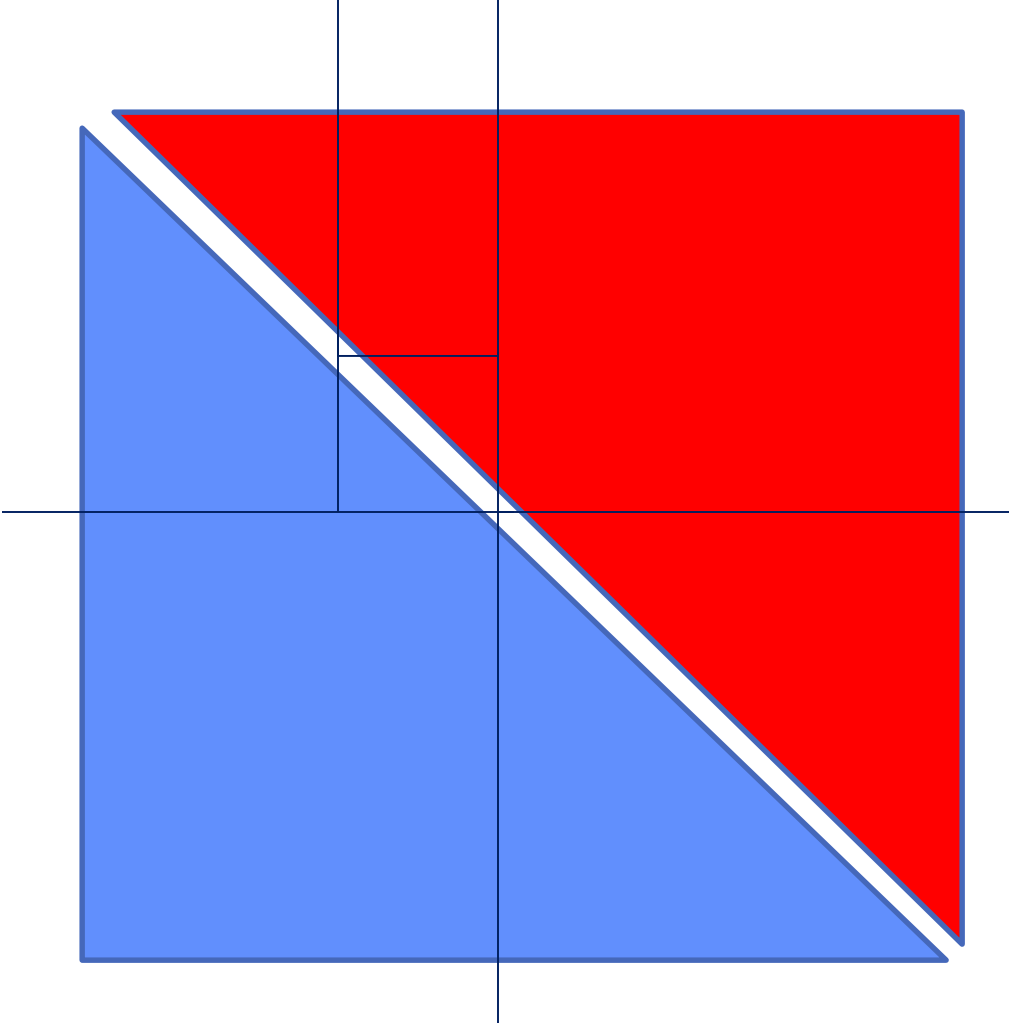
## Classifier Bias – Decision Tree or Linear Perceptron?



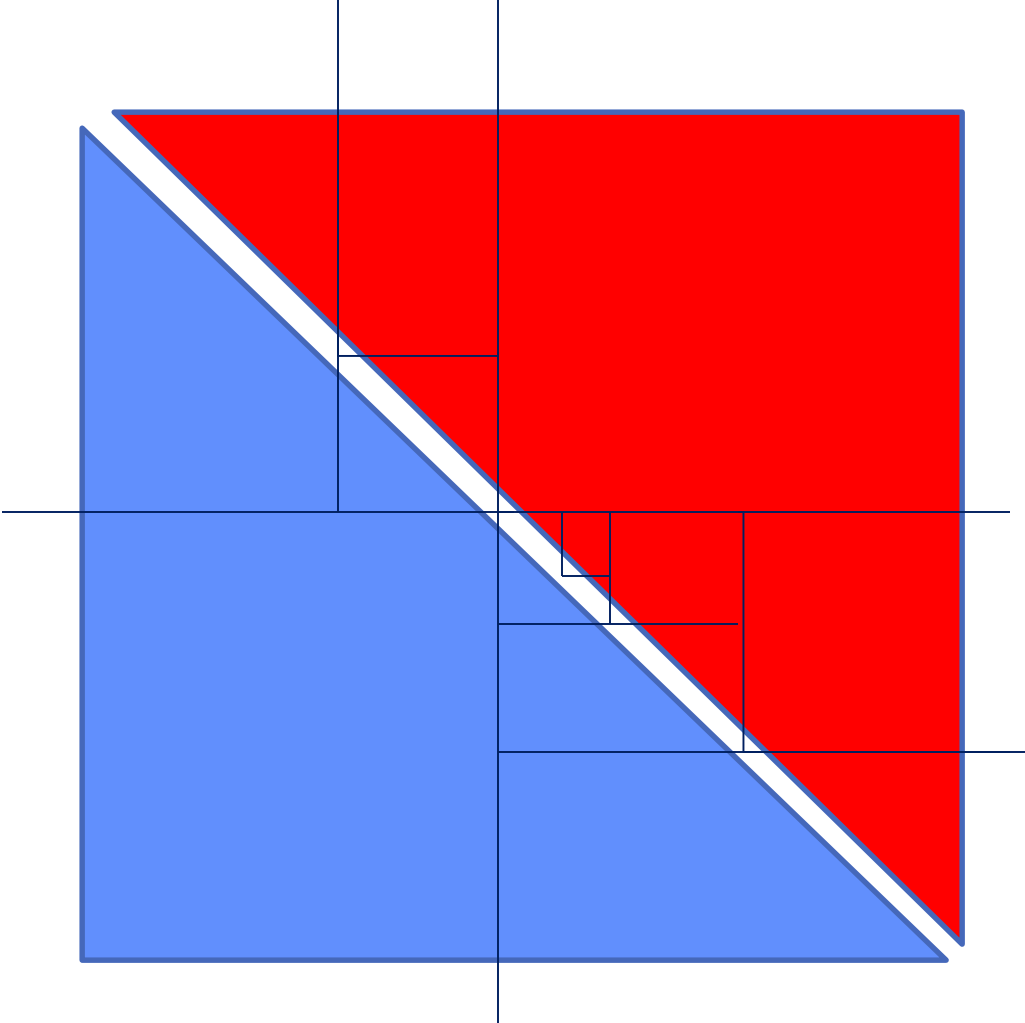
## Classifier Bias – Decision Tree or Linear Perceptron?



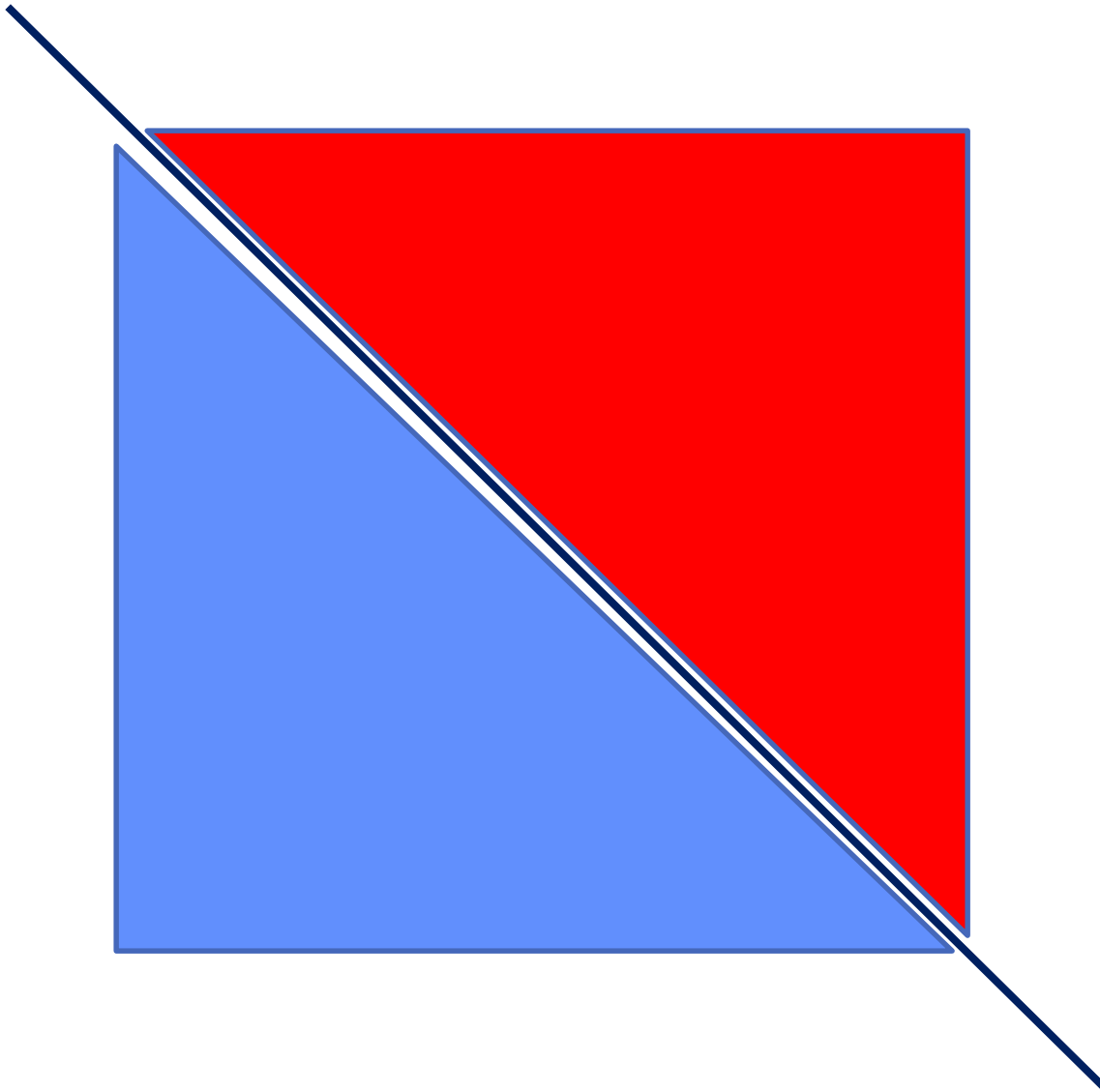
# Classifier Bias – Decision Tree or Linear Perceptron?



# Classifier Bias – Decision Tree or Linear Perceptron?

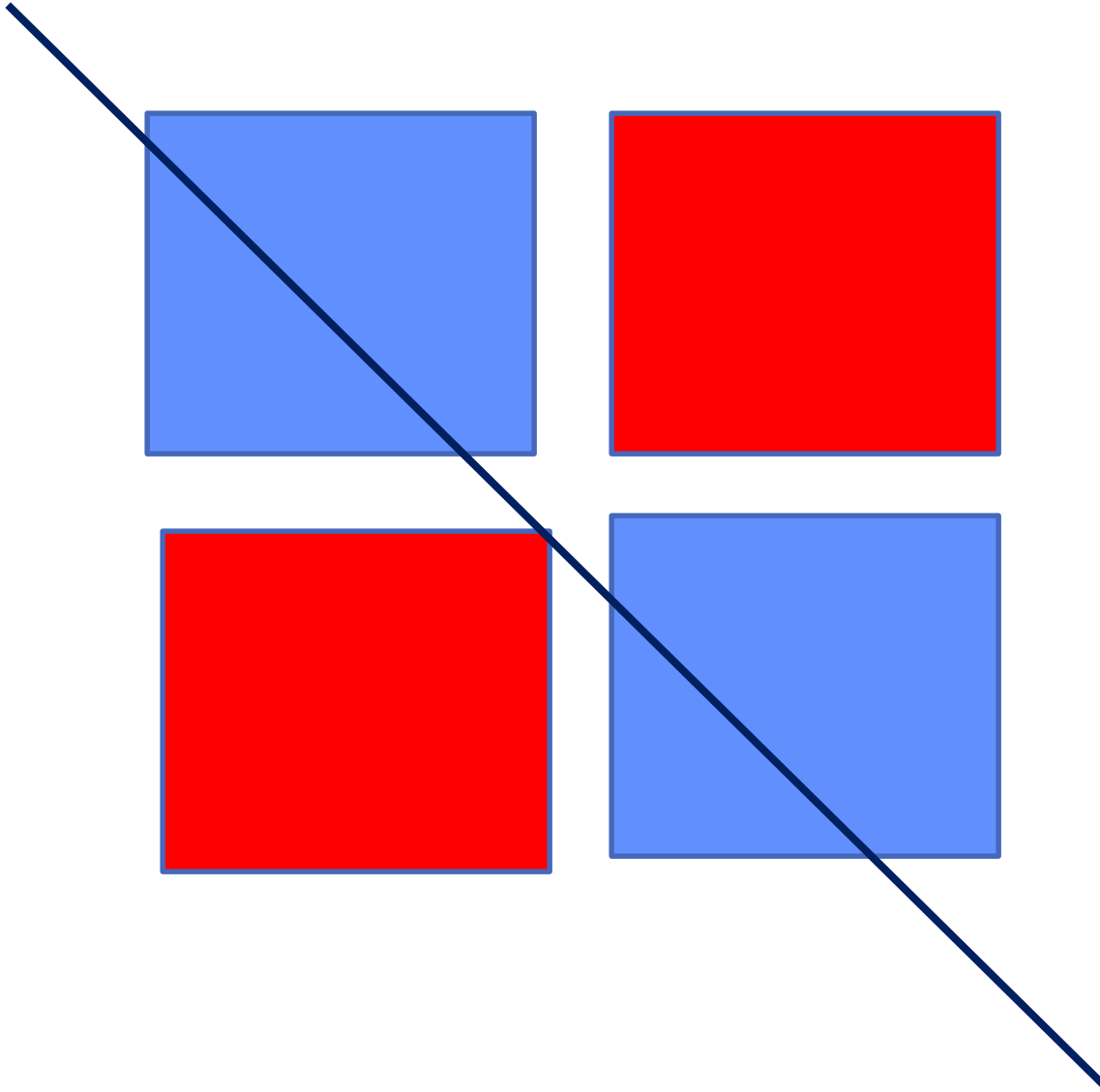


## Classifier Bias – Decision Tree or Linear Perceptron?

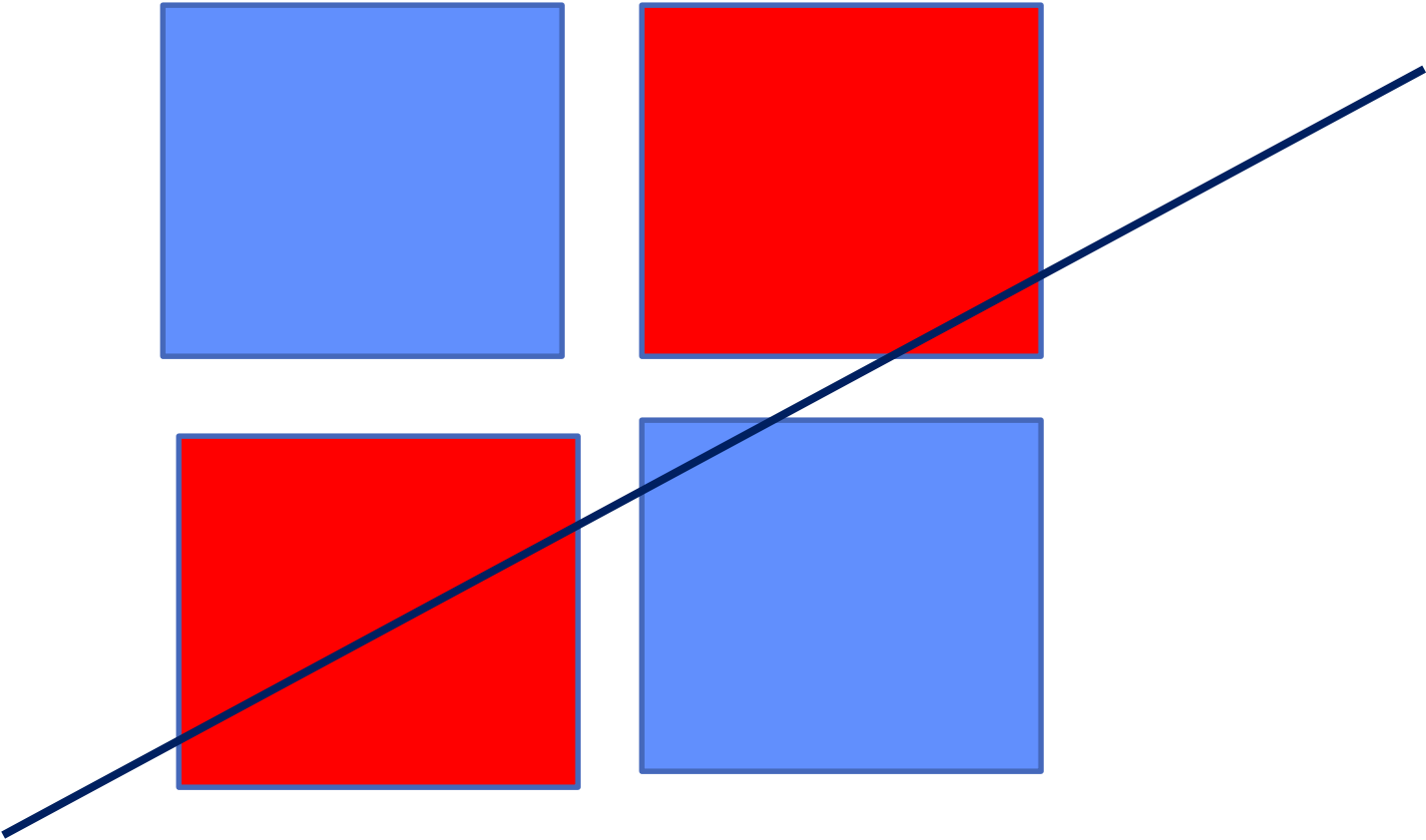




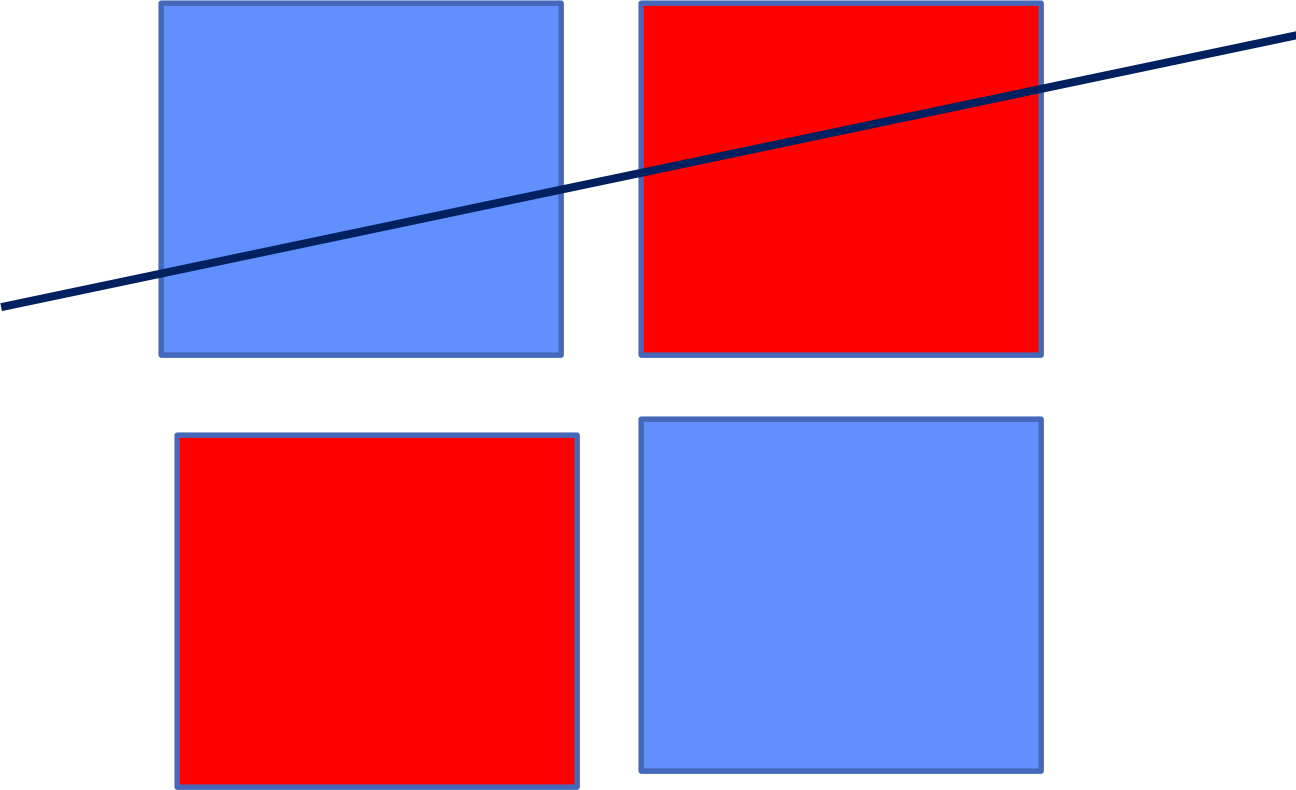
## Classifier Bias – Decision Tree or Linear Perceptron?



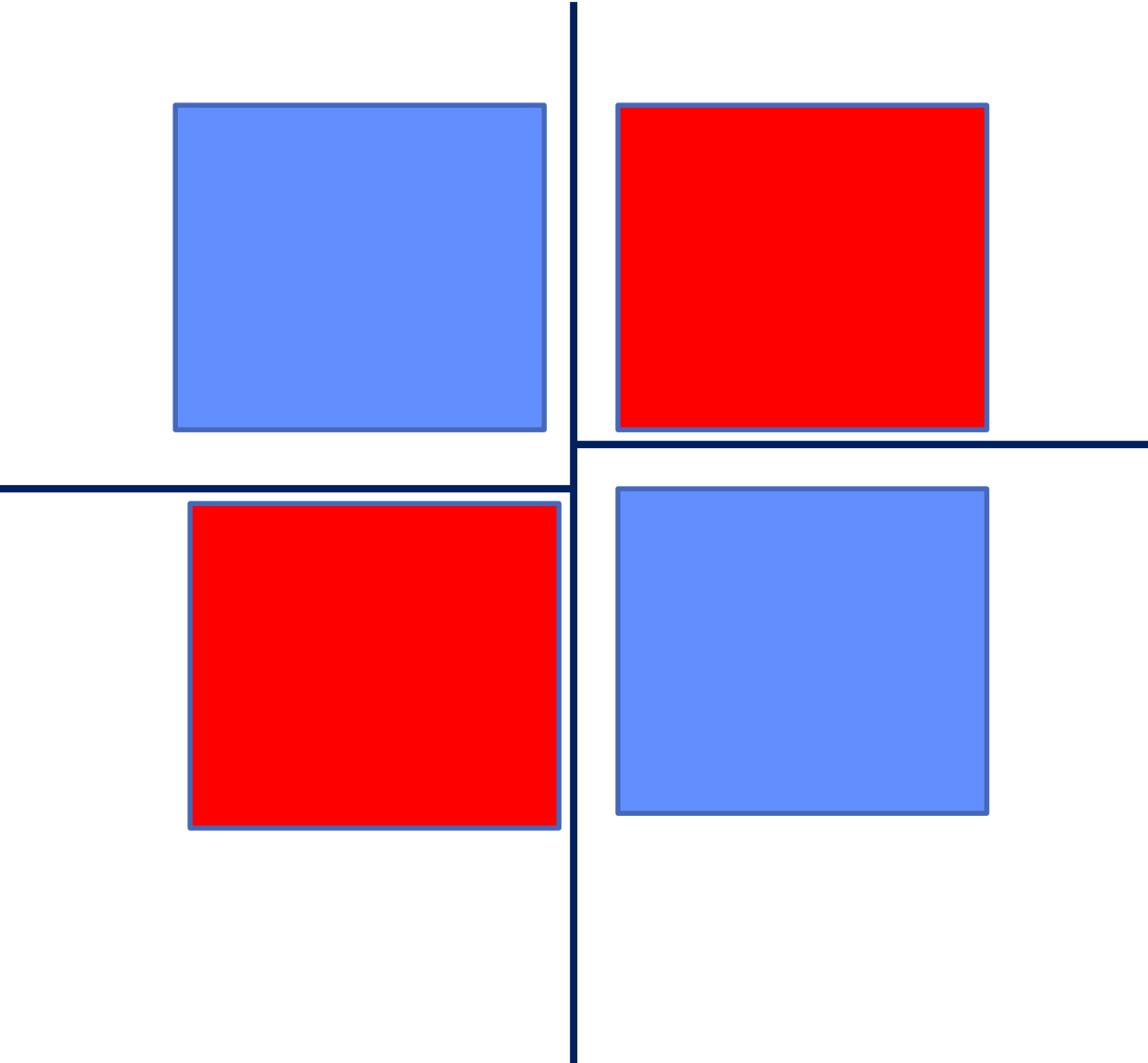
# Classifier Bias – Decision Tree or Linear Perceptron?



# Classifier Bias – Decision Tree or Linear Perceptron?



# Classifier Bias – Decision Tree or Linear Perceptron?



# Summary

- Learning
  - Given a training data set, a class of models, and an error function, this is essentially a search or optimization problem
- Different approaches to learning
  - Divide-and-conquer: decision trees
  - Global decision boundary learning: perceptrons
  - Constructing classifiers incrementally: boosting
- Learning to recognize faces
  - Viola-Jones algorithm: state-of-the-art face detector, entirely learned from data, using boosting+decision-stumps