Lecture 15
Machine Learning
KNN, Minimum Distance, Perceptron

The Major Machine Learning Tasks

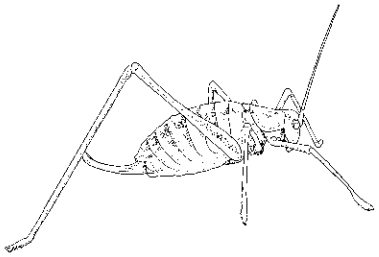
- Classification
- Clustering

Most of the other tasks (for example, outlier discovery or anomaly detection) make heavy use of one or more of the above.

So in this tutorial we will focus most of our energy on the above, starting with…

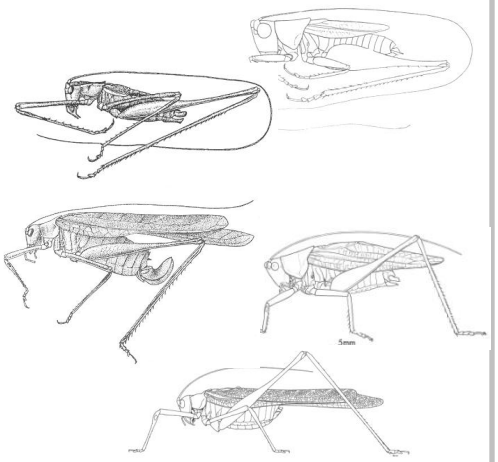
The Classification Problem
(informal definition)

Given a collection of annotated data. In this case 5 instances of **Katydids** and five of **Grasshoppers**, decide what type of insect the unlabeled example is.

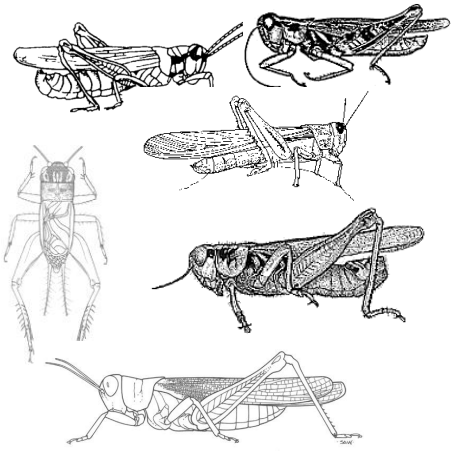


Katydid or **Grasshopper**?

Katydids



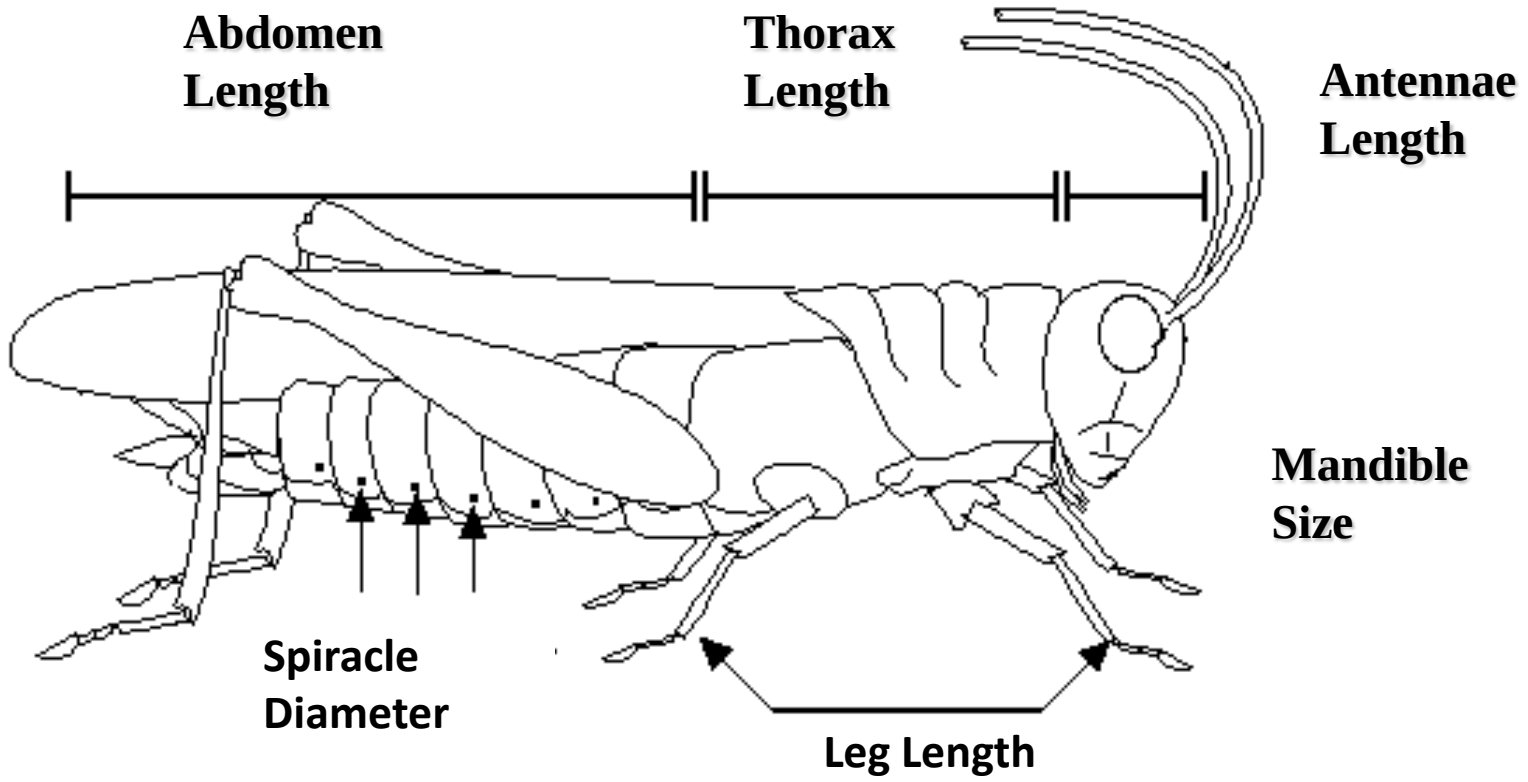
Grasshoppers



For any domain of interest, we can measure *features*

Color {Green, Brown, Gray, Other}

Has Wings?



We can store features in a database.

The classification problem can now be expressed as:

• Given a training database (**My_Collection**), predict the **class** label of a previously unseen instance

My_Collection

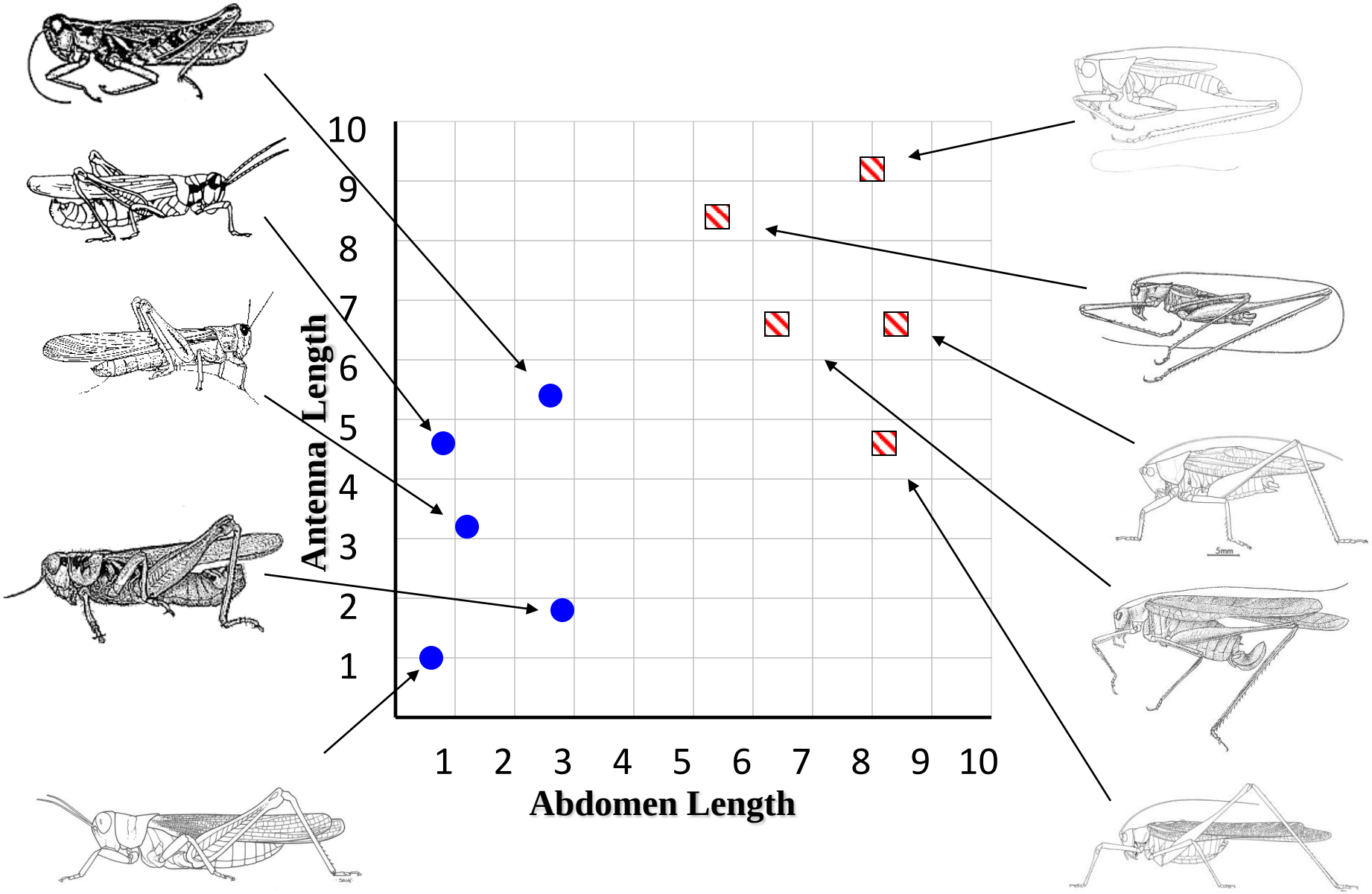
Insect ID	Abdomen Length	Antennae Length	Insect Class
1	2.7	5.5	**Grasshopper**
2	8.0	9.1	**Katydid**
3	0.9	4.7	**Grasshopper**
4	1.1	3.1	**Grasshopper**
5	5.4	8.5	**Katydid**
6	2.9	1.9	**Grasshopper**
7	6.1	6.6	**Katydid**
8	0.5	1.0	**Grasshopper**
9	8.3	6.6	**Katydid**
10	8.1	4.7	**Katydids**

previously unseen instance =

11	5.1	7.0	???????

Grasshoppers

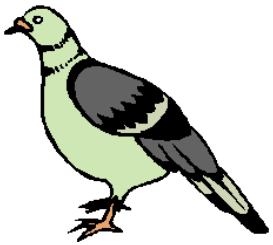
Katydids



We will return to the previous slide in two minutes. In the meantime, we are going to play a quick game.

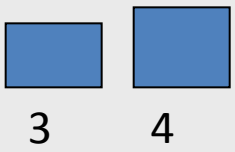
I am going to show you some classification problems which were shown to pigeons!

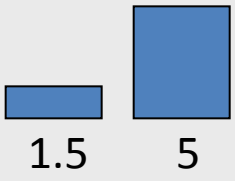
Let us see if you are as smart as a pigeon!

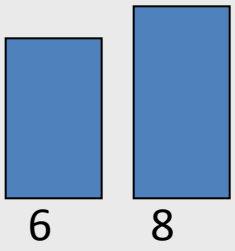


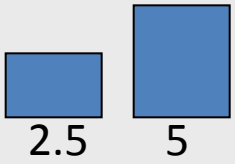
Pigeon Problem 1

Examples of class A

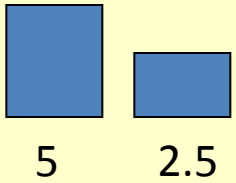


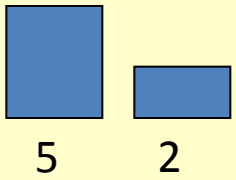


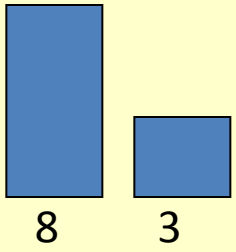


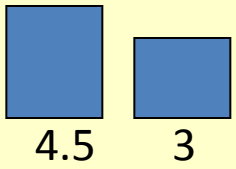


Examples of class B



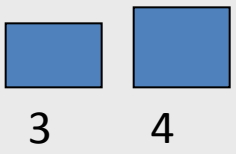


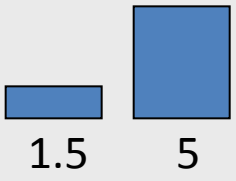


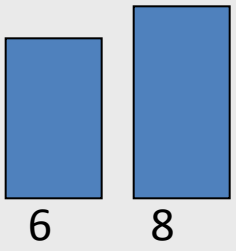


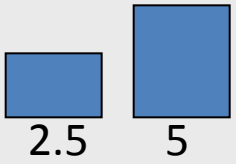
Pigeon Problem 1

Examples of class A

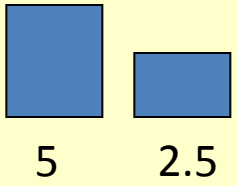


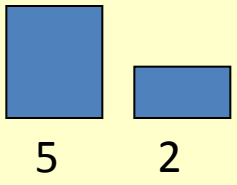


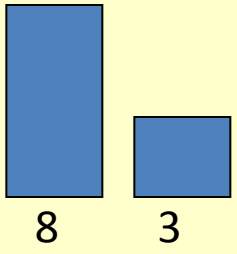


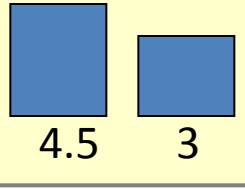


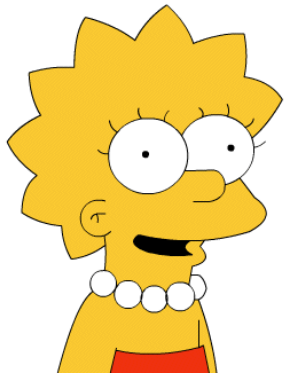
Examples of class B



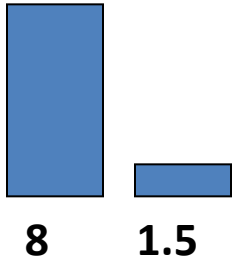




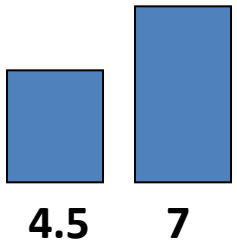




What class is this object?

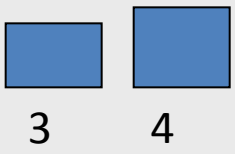


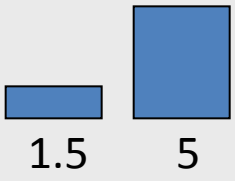
What about this one, A or B?

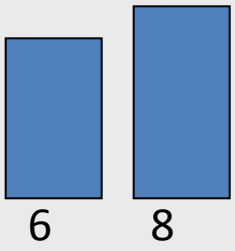


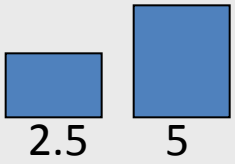
Pigeon Problem 1

Examples of class A

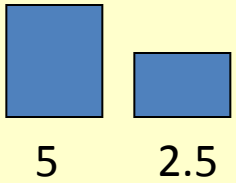


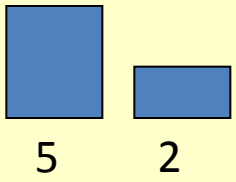


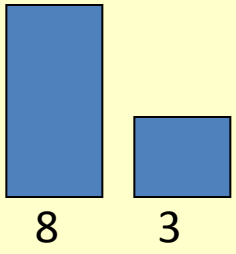


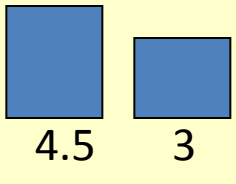


Examples of class B

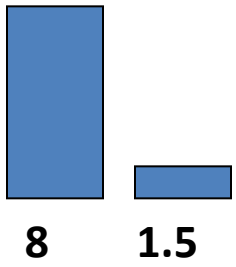








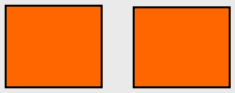
This is a B!

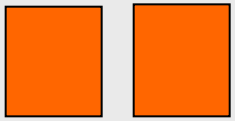


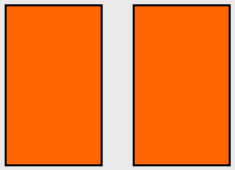
Here is the rule.
If the left bar is smaller than the right bar, it is an A, otherwise it is a B.

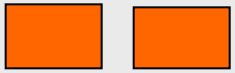
Pigeon Problem 2

Examples of class A

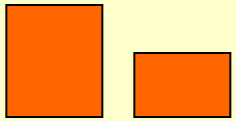


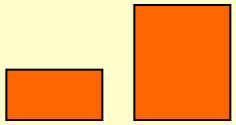


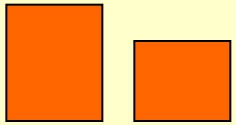


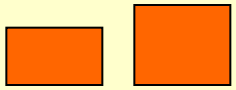


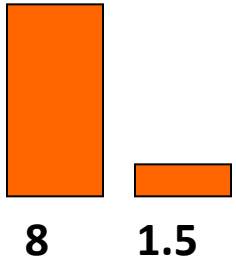
Examples of class B

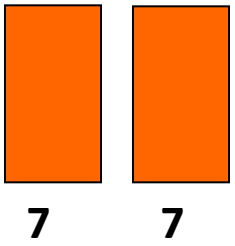






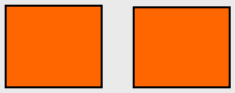


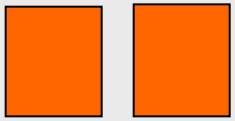


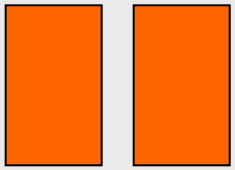


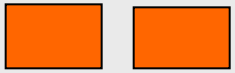
Pigeon Problem 2

Examples of class A

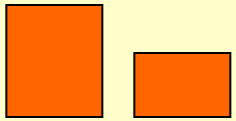

4 4

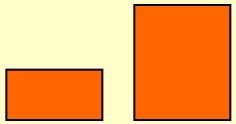

5 5

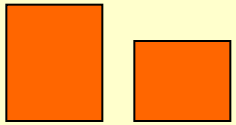

6 6

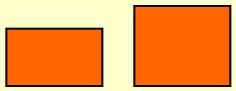

3 3

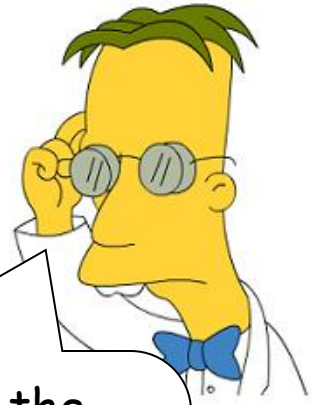
Examples of class B


5 2.5


2 5

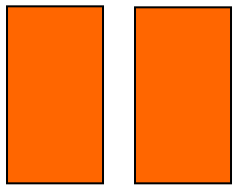

5 3


2.5 3



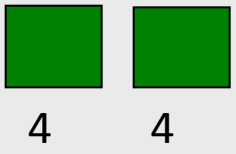
The rule is as follows, if the two bars are equal sizes, it is an A. Otherwise it is a B.

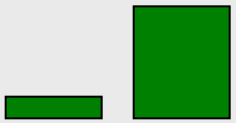
So this one is an A.

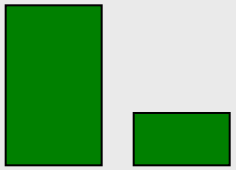

7 7

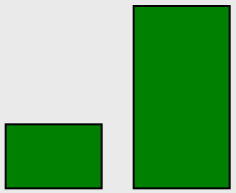
Pigeon Problem 3

Examples of class A

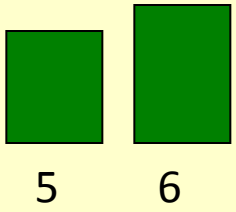


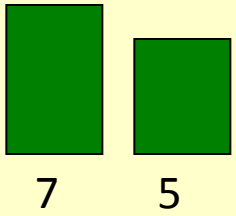


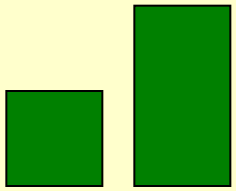


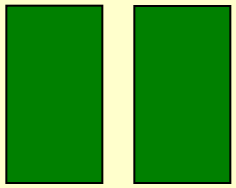


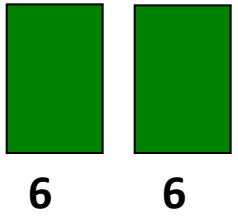
Examples of class B







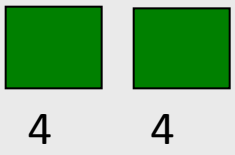


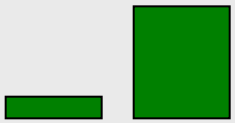


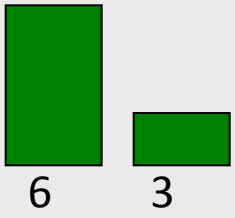
This one is really hard!
What is this, A or B?

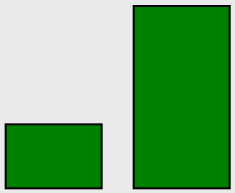
Pigeon Problem 3

Examples of class A

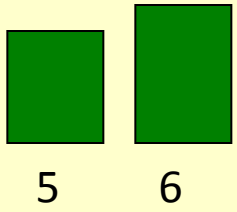


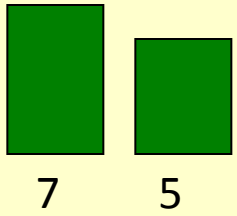


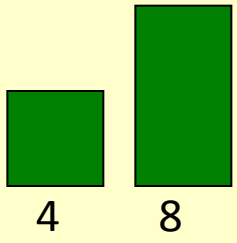


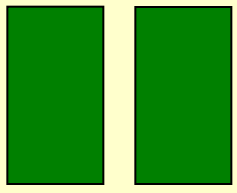


Examples of class B

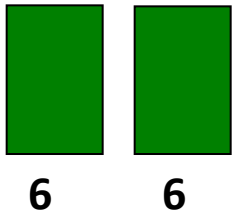








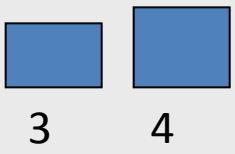
It is a B!

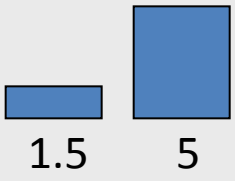


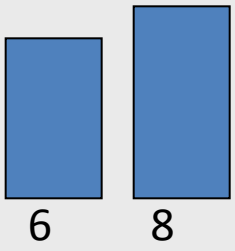
The rule is as follows, if the sum of the two bars is less than or equal to 10, it is an A. Otherwise it is a B.

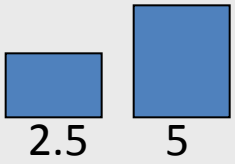
Pigeon Problem 1

Examples of class A

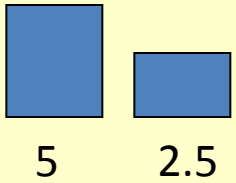


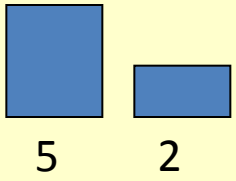


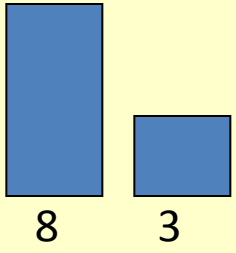


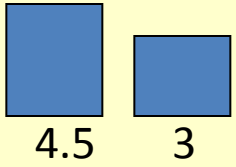


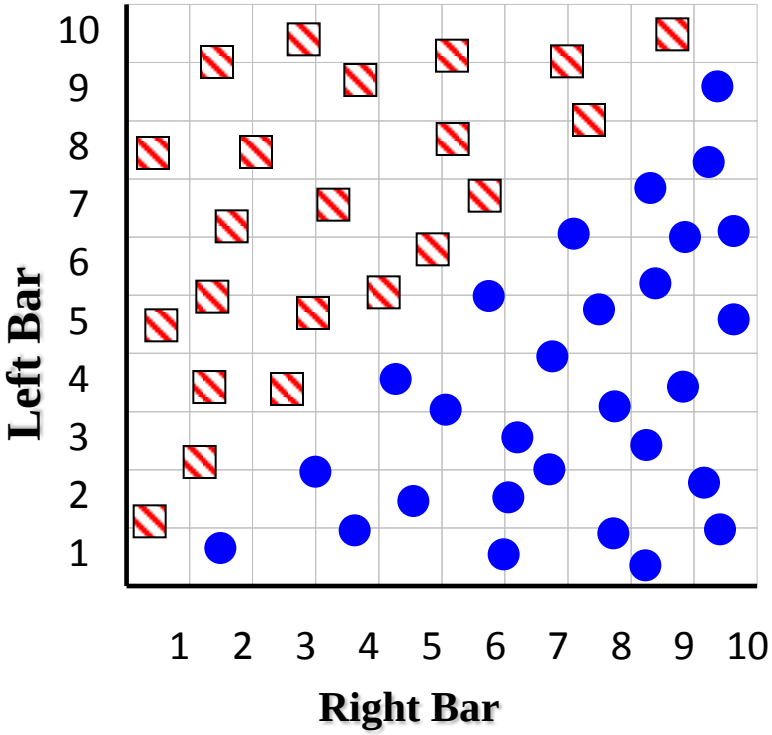
Examples of class B







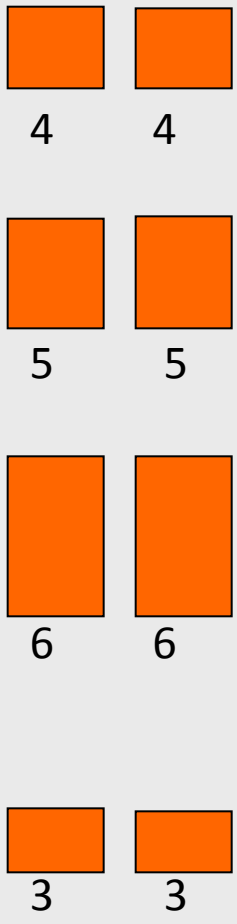




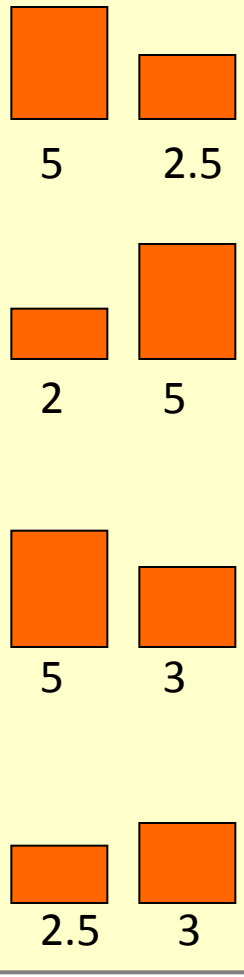
Here is the rule again.
If the left bar is smaller than the right bar, it is an A, otherwise it is a B.

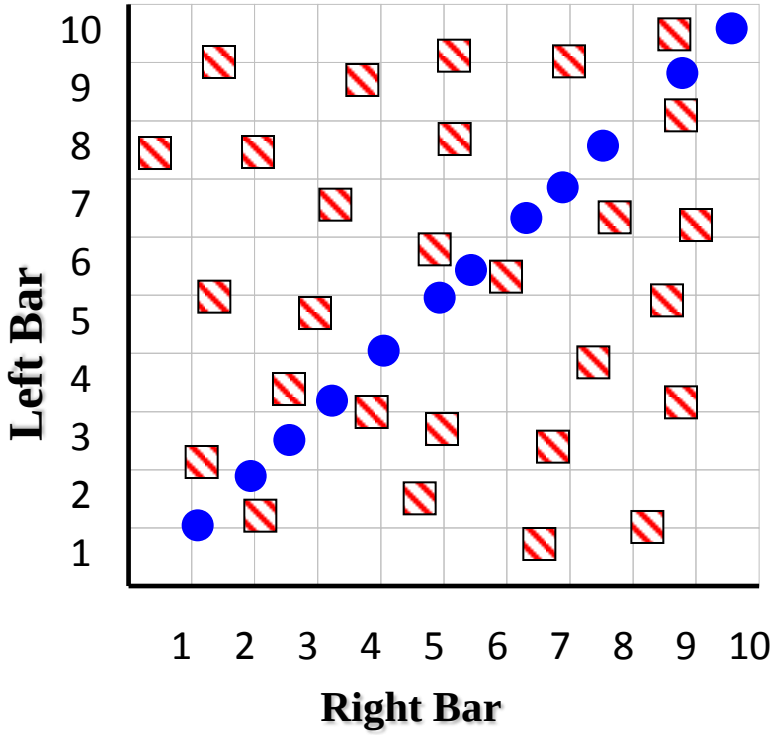
Pigeon Problem 2

Examples of class A



Examples of class B

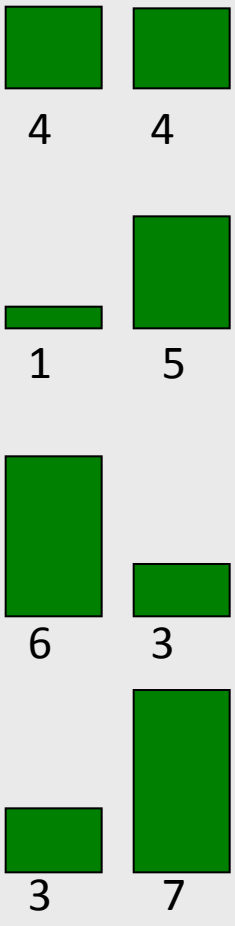




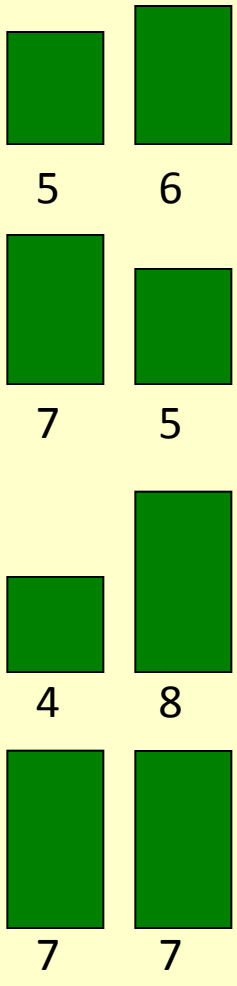
Let me look it up… here it is.. the rule is, if the two bars are equal sizes, it is an A. Otherwise it is a B.

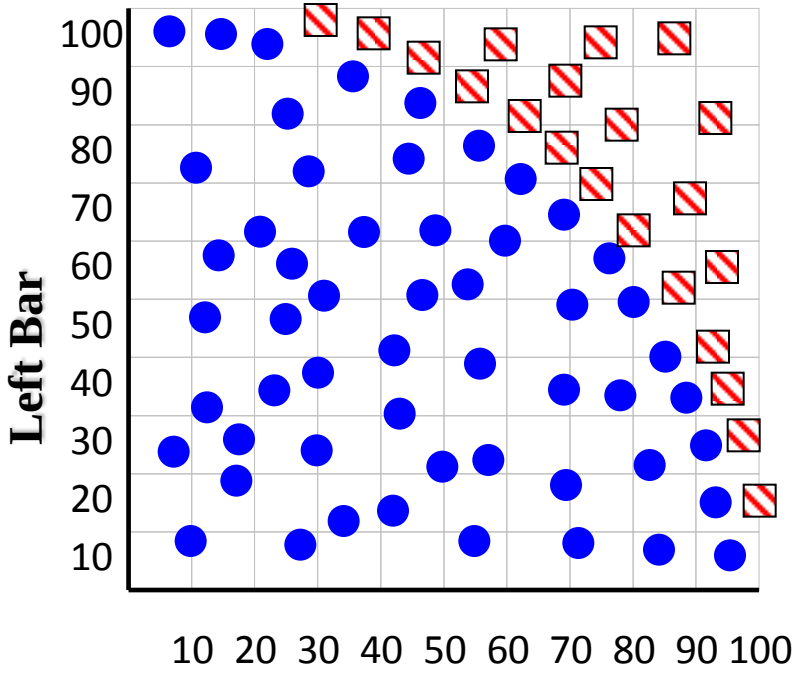
Pigeon Problem 3

Examples of class A



Examples of class B

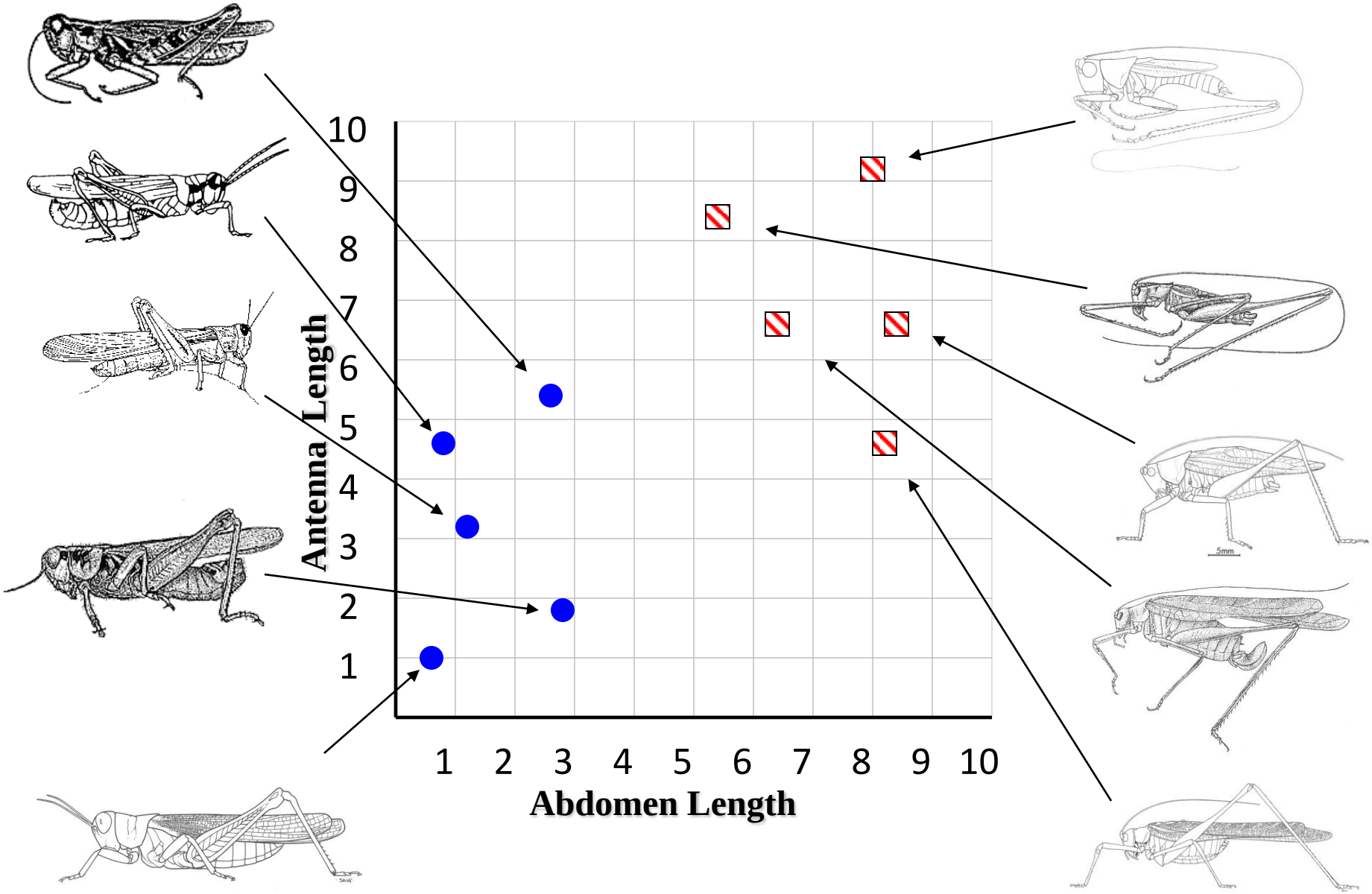




The rule again:
if the square of the sum of the two bars is less than or equal to 100, it is an A. Otherwise it is a B.

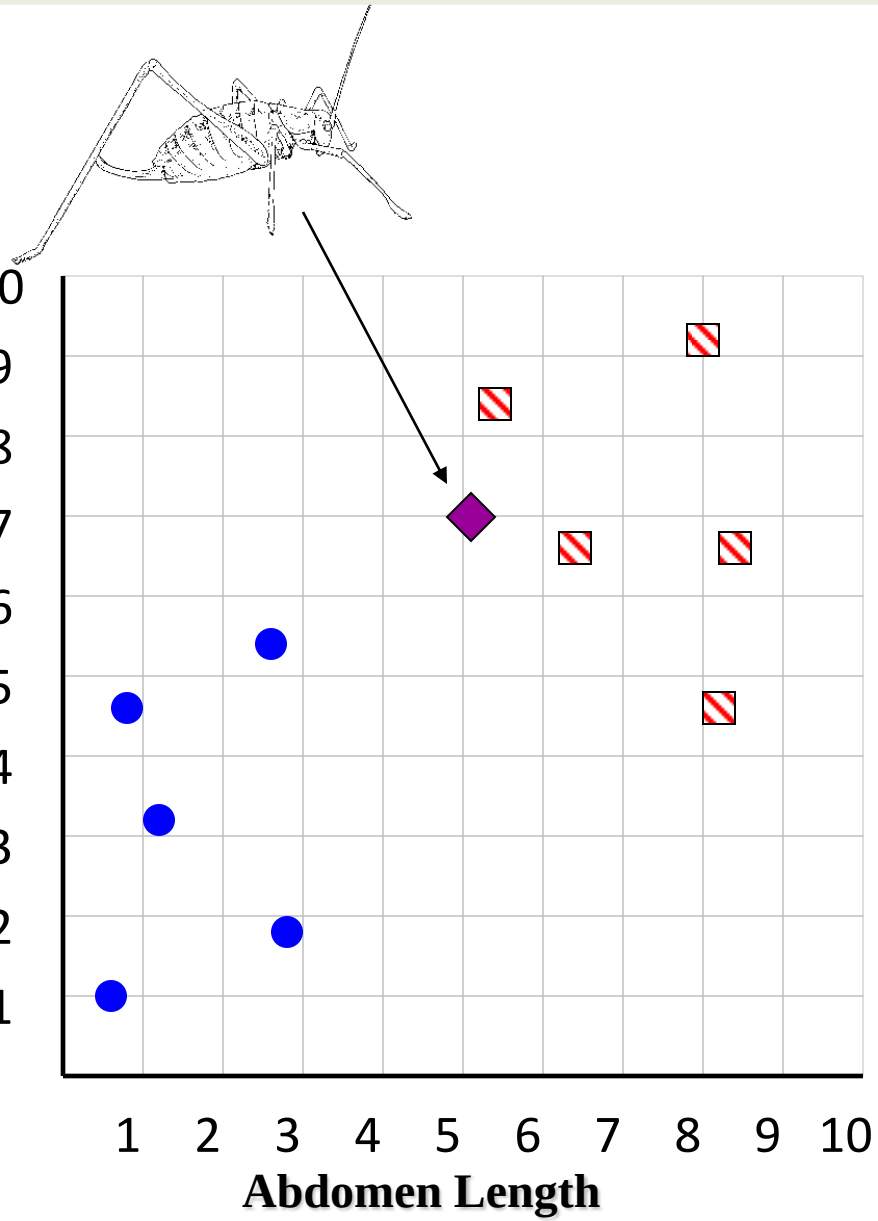
Grasshoppers

Katydids



previously unseen instance =

11	**5.1**	**7.0**	**???????**



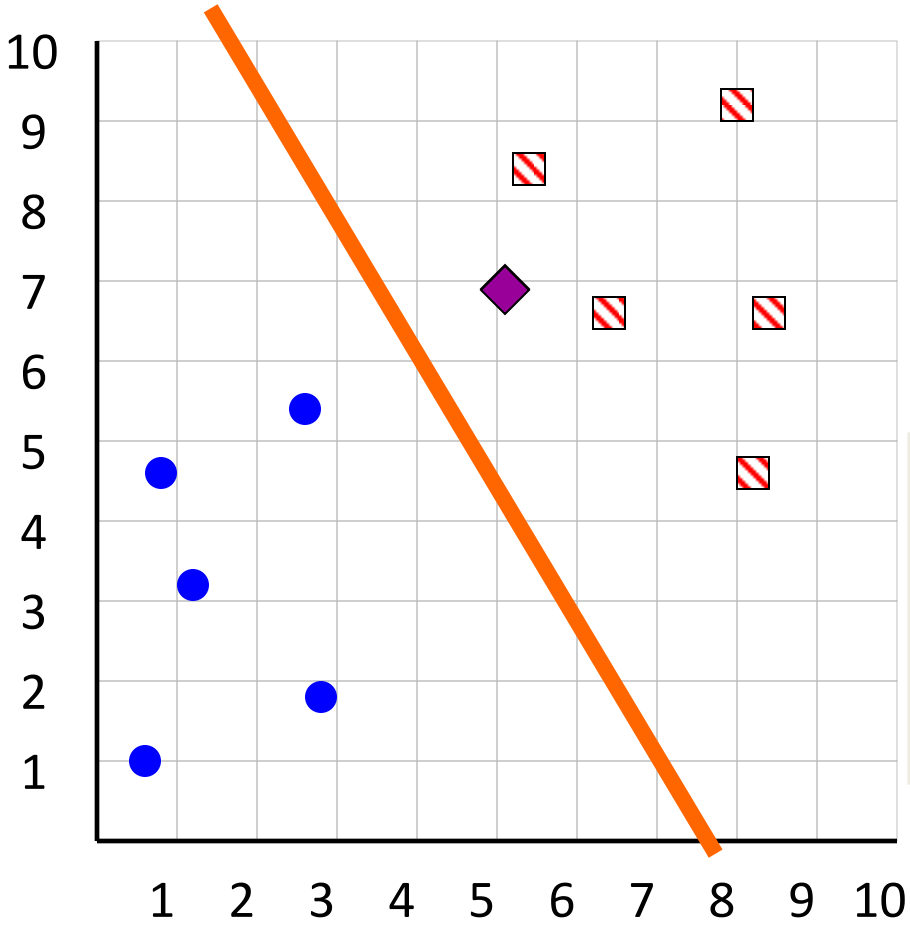
• We can "project" the **previously unseen instance** into the same space as the database.

• We have now abstracted away the details of our particular problem. It will be much easier to talk about points in space.

Katydids

Grasshoppers

Simple Linear Classifier



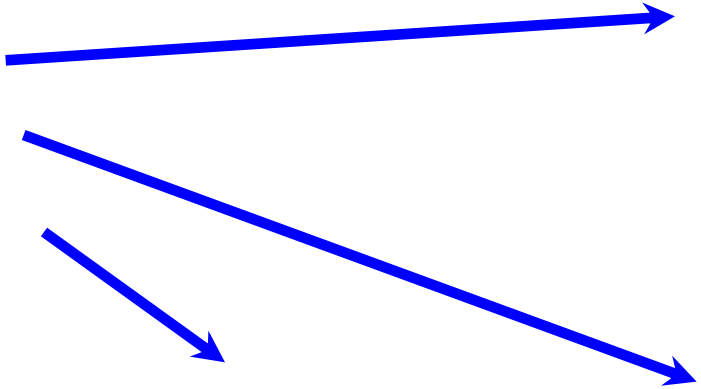
R.A. Fisher
1890-1962

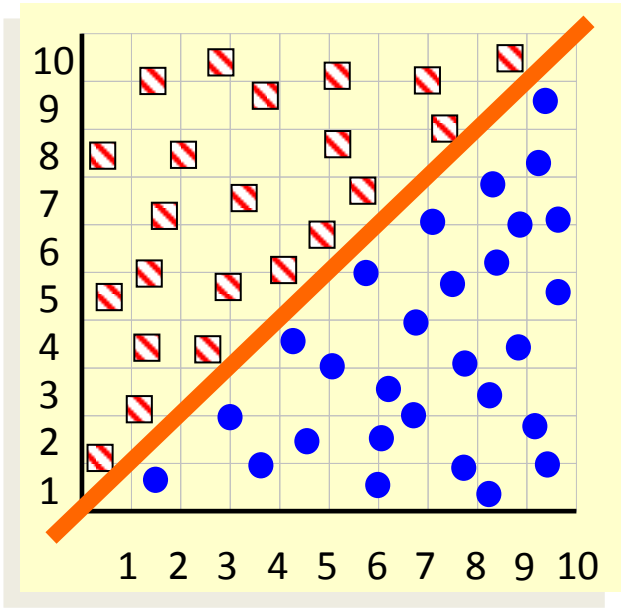
If previously unseen instance above the line
then
class is **Katydid**
else
class is **Grasshopper**

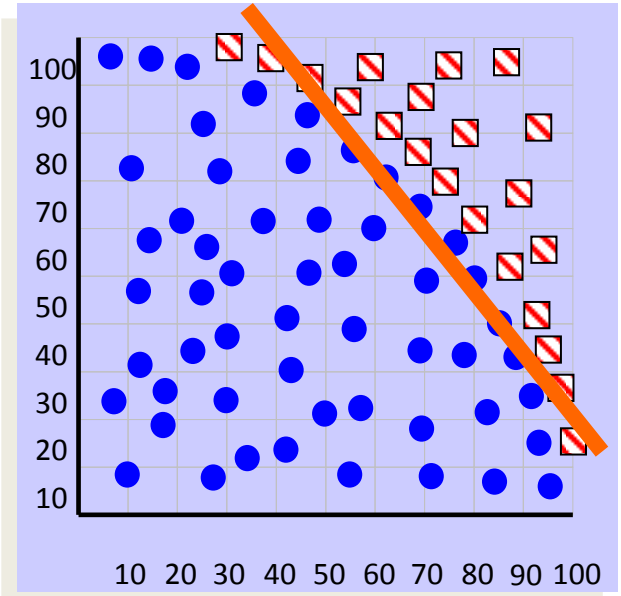
Katydids
Grasshoppers

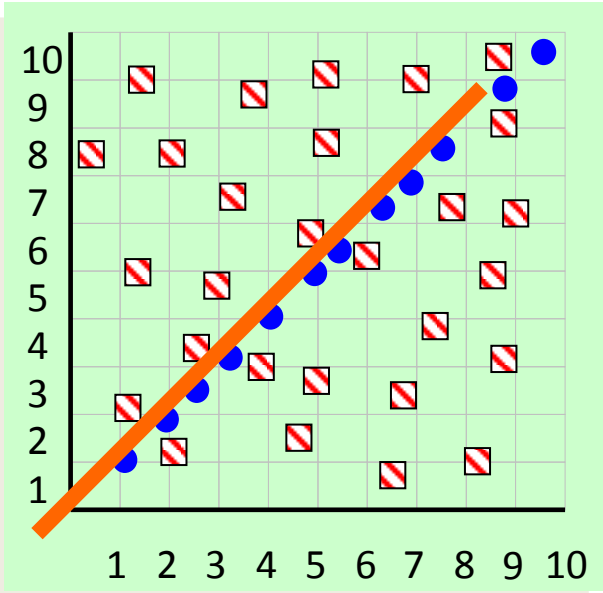
Which of the "Pigeon Problems" can be solved by the Simple Linear Classifier?

1) Perfect
2) Useless
3) Pretty Good









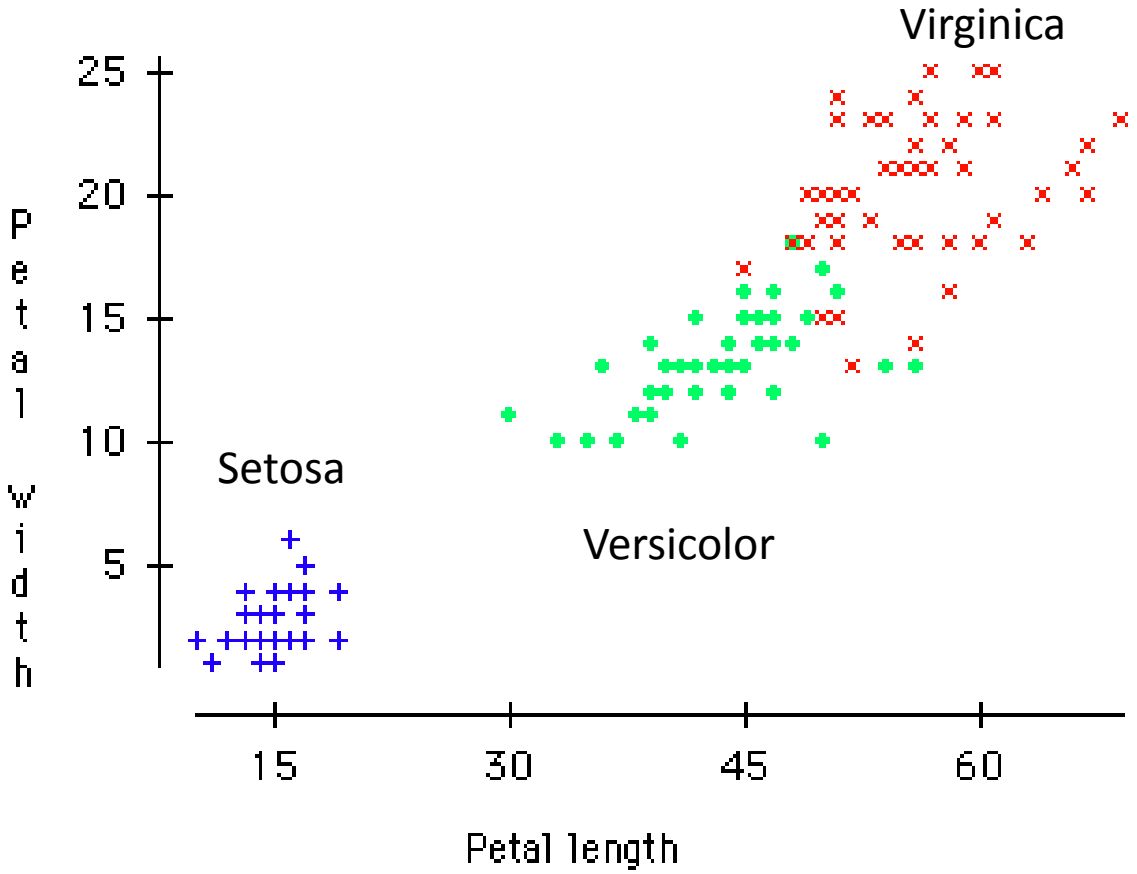
Problems that can be solved by a linear classifier are call **linearly separable**.

A Famous Problem

R. A. Fisher's Iris Dataset.

- 3 classes
- 50 of each class

The task is to classify Iris plants into one of 3 varieties using the Petal Length and Petal Width.

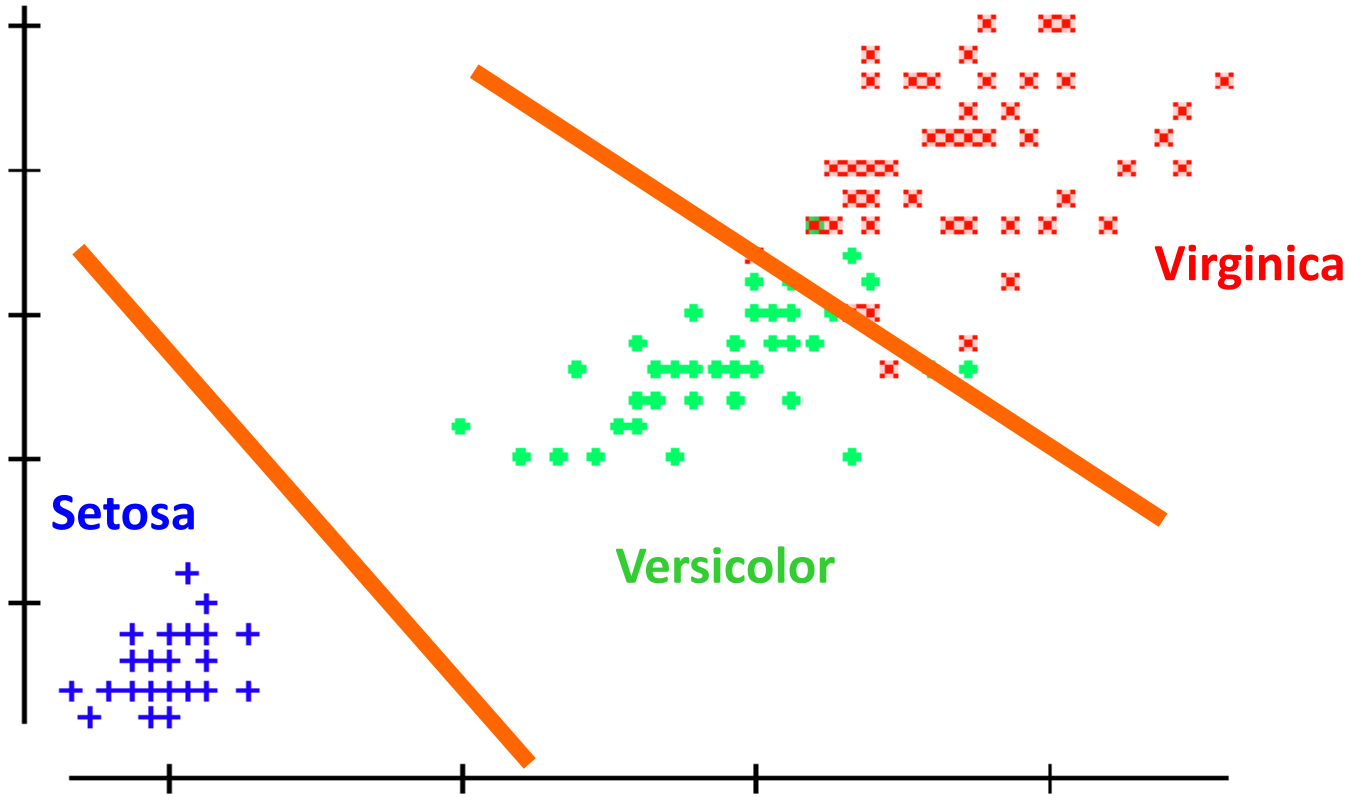


Iris Setosa

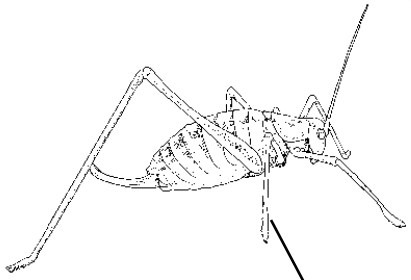
Iris Versicolor

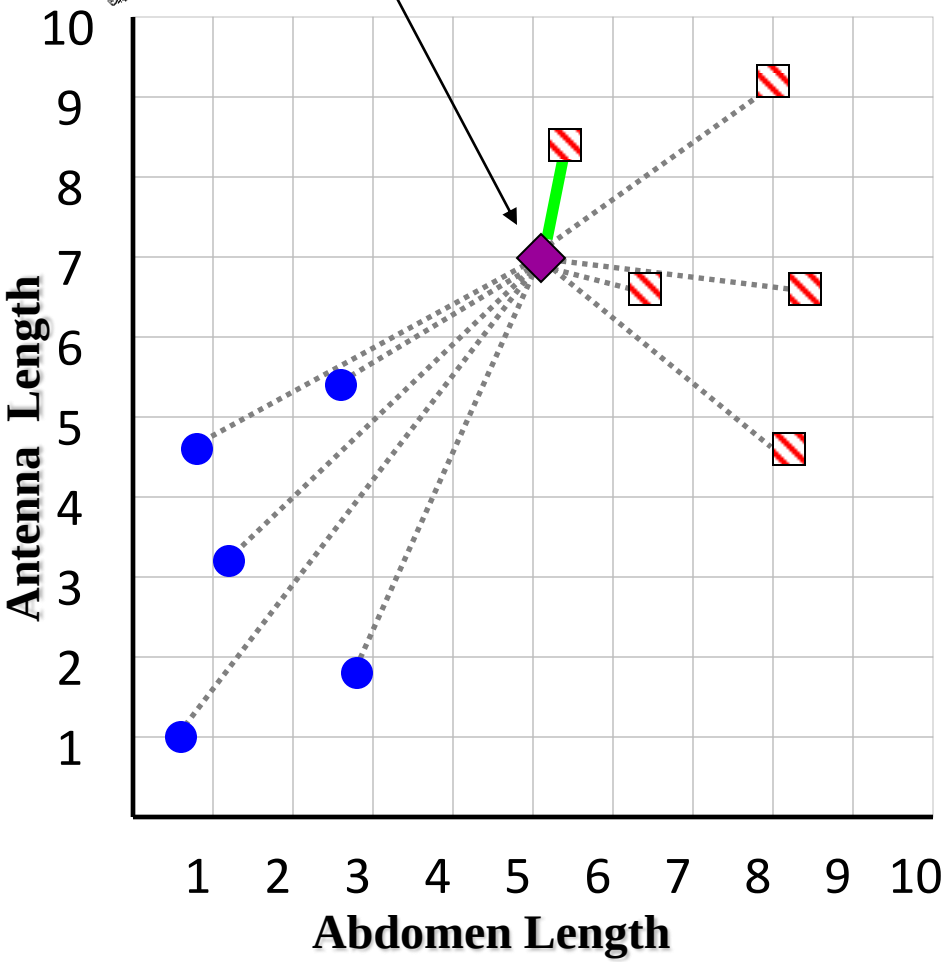
Iris Virginica

We can generalize the piecewise linear classifier to N classes, by fitting N-1 lines. In this case we first learned the line to (perfectly) discriminate between **Setosa** and **Virginica**/**Versicolor**, then we learned to approximately discriminate between **Virginica** and **Versicolor**.



Nearest Neighbor Classifier





If the **nearest** instance to the previously unseen instance **is a Katydid**

class is **Katydid**

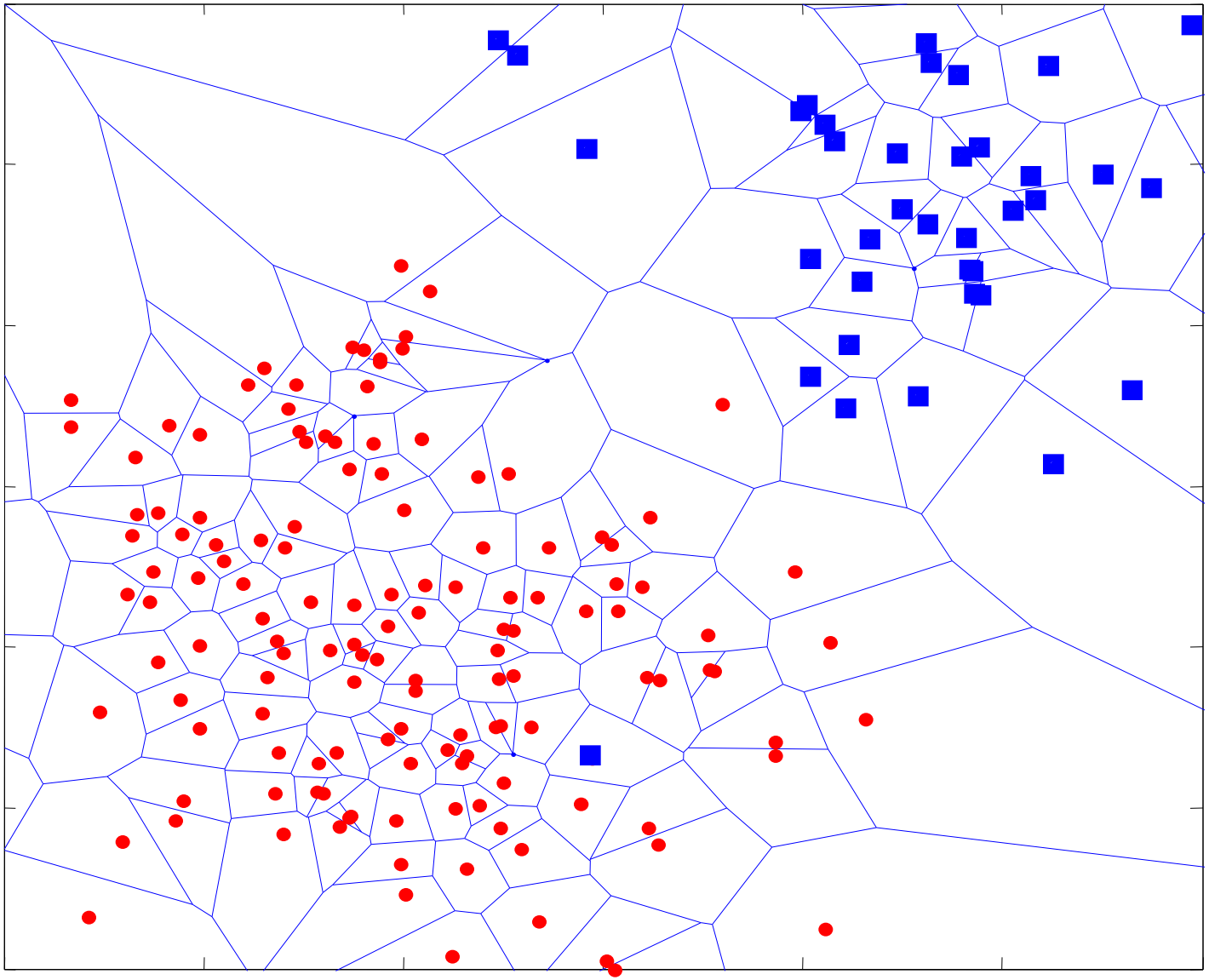
else

class is **Grasshopper**

Katydids

Grasshoppers

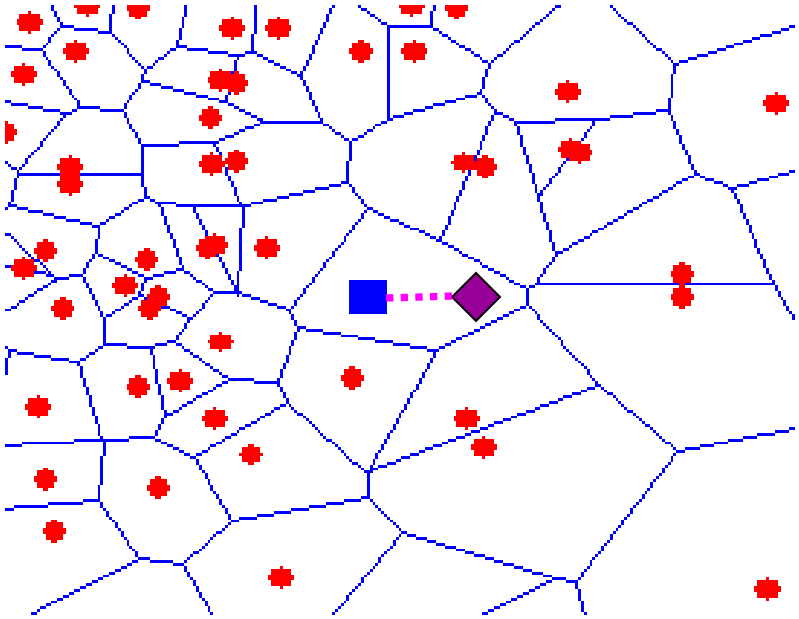
The nearest neighbor algorithm is sensitive to outliers…



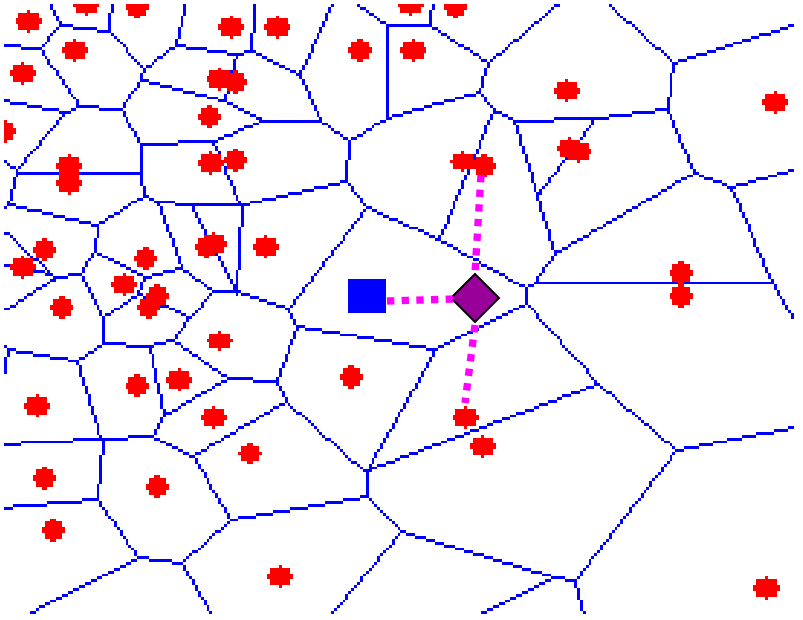
The solution is to…

We can generalize the nearest neighbor algorithm to the K- nearest neighbor (KNN) algorithm.

We measure the distance to the nearest K instances, and let them vote. K is typically chosen to be an odd number.



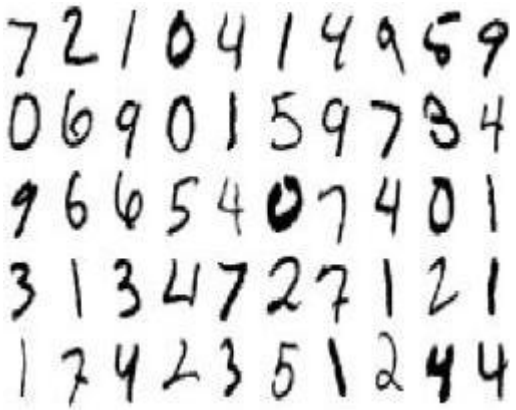
K = 1



K = 3

KNN

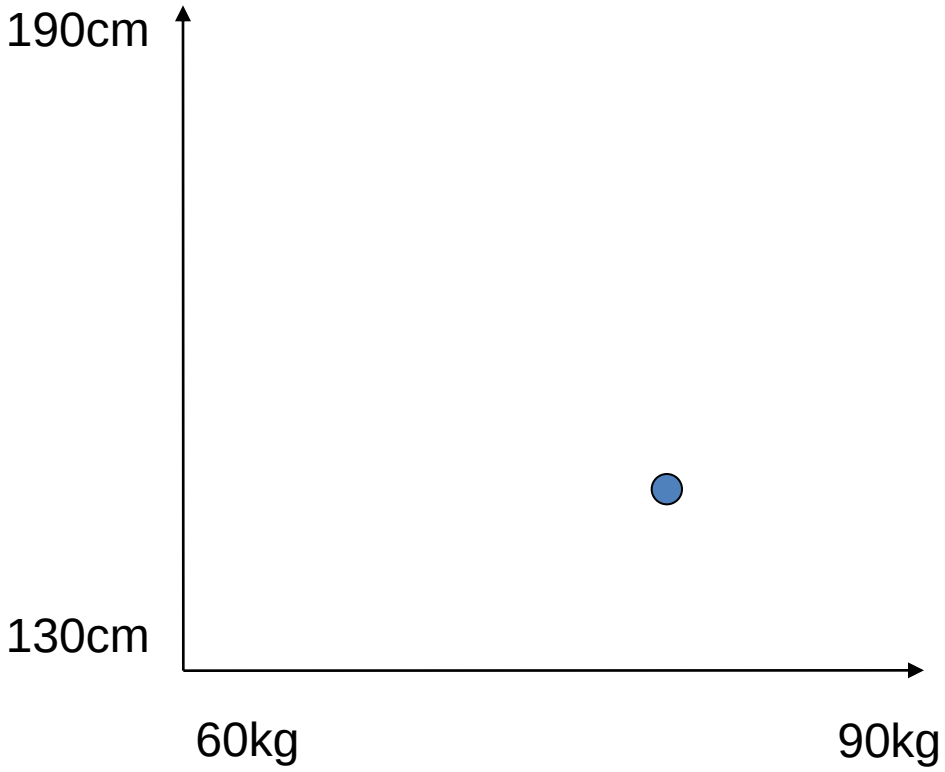
- Why recognising rugby players is (almost) the same problem as *handwriting recognition*



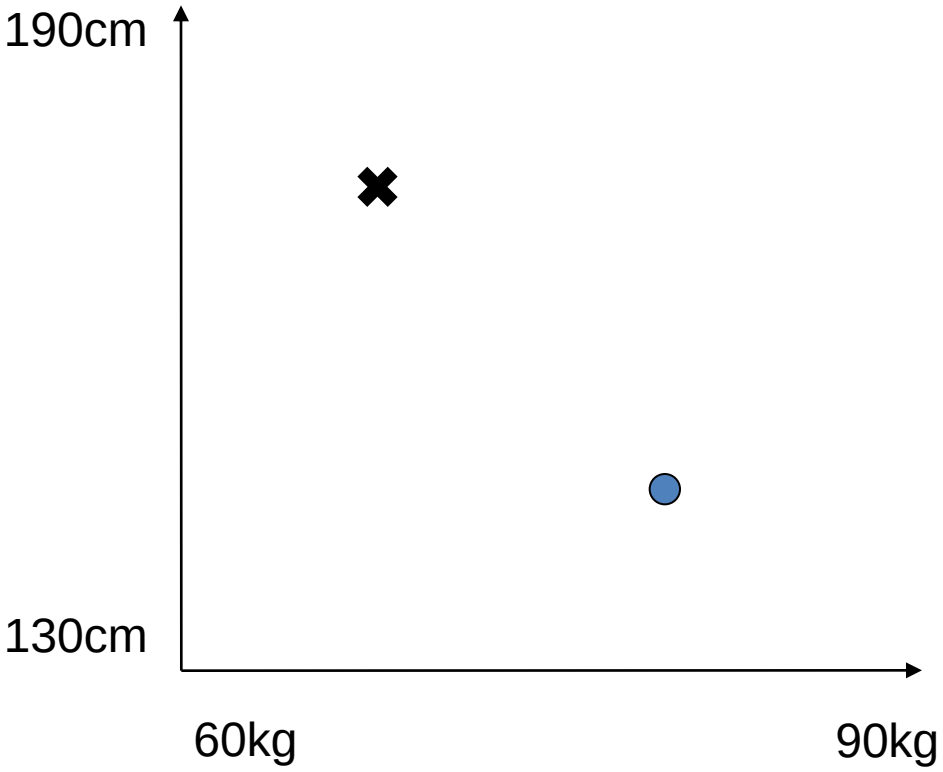
Can we LEARN to recognise a rugby player?

What are the “features” of a rugby player?

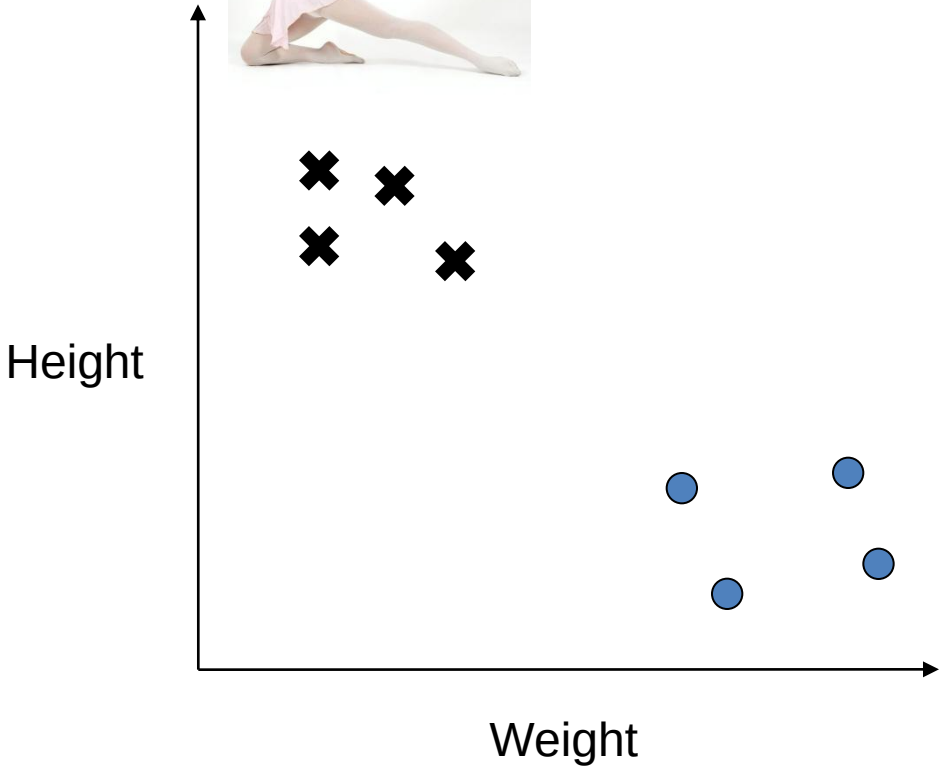
Rugby players = short + heavy?



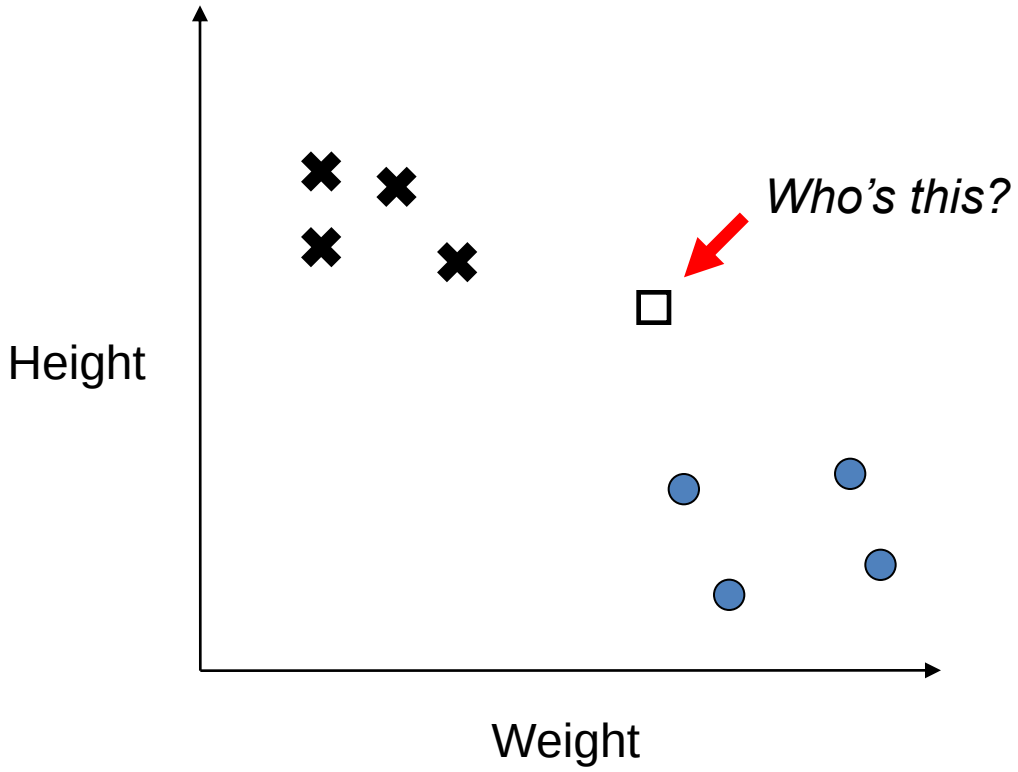
Ballet dancers = tall + skinny?



Rugby players “cluster” separately in the space.

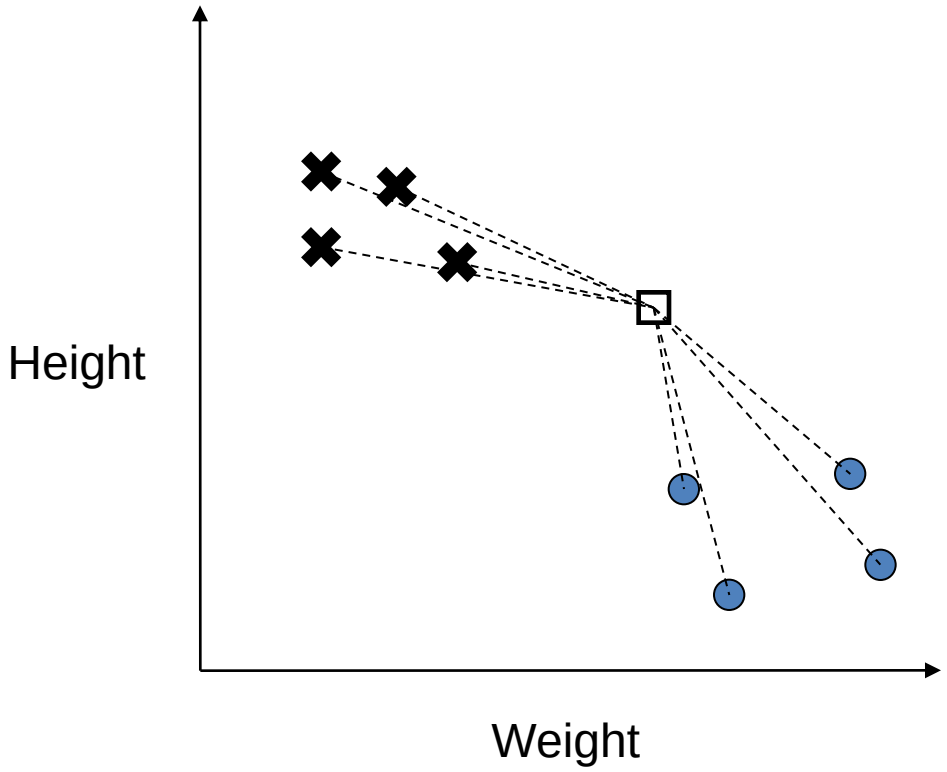


The K-Nearest Neighbour Algorithm



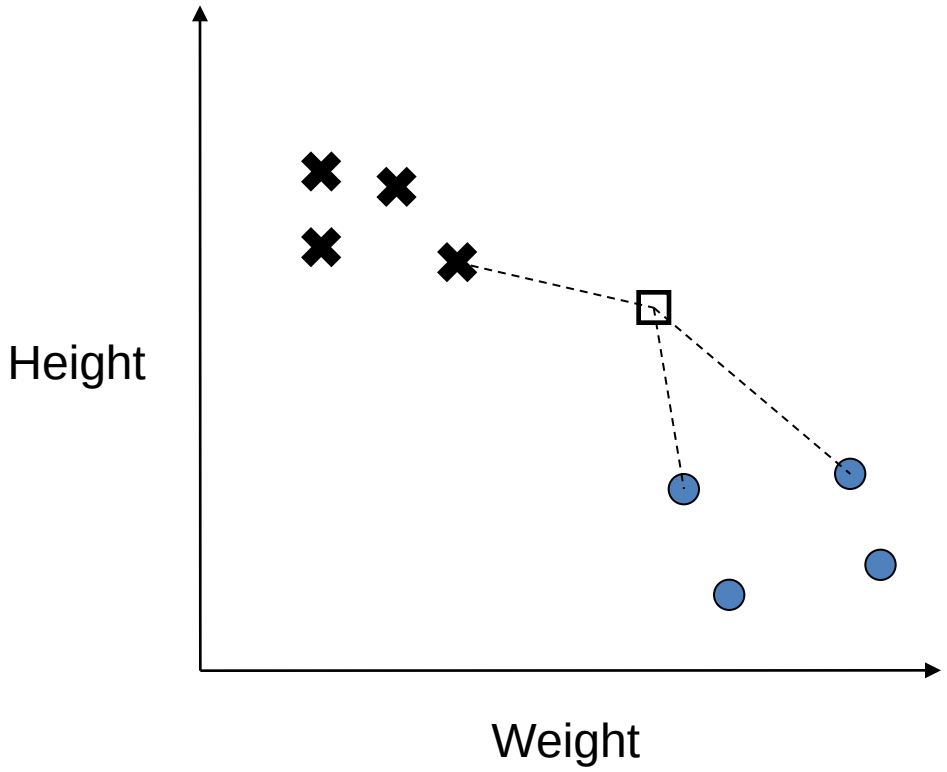
The K-Nearest Neighbour Algorithm

1. *Measure distance to all points*



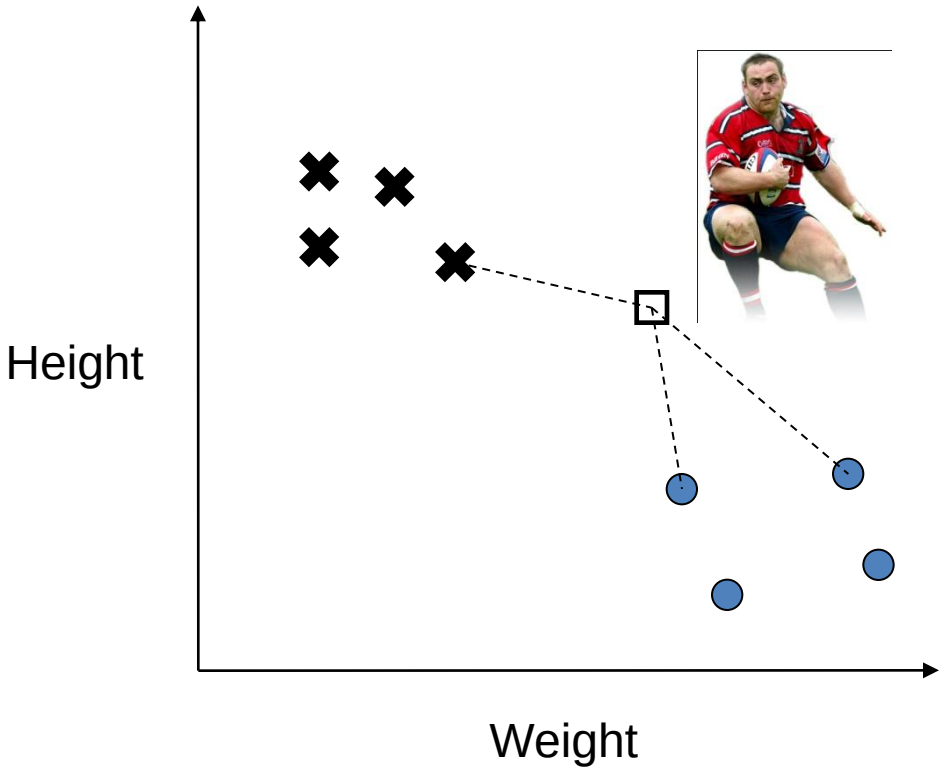
The K-Nearest Neighbour Algorithm

1. *Measure distance to all points*
2. *Find closest "k" points* ← **(here k=3, but it could be more)**



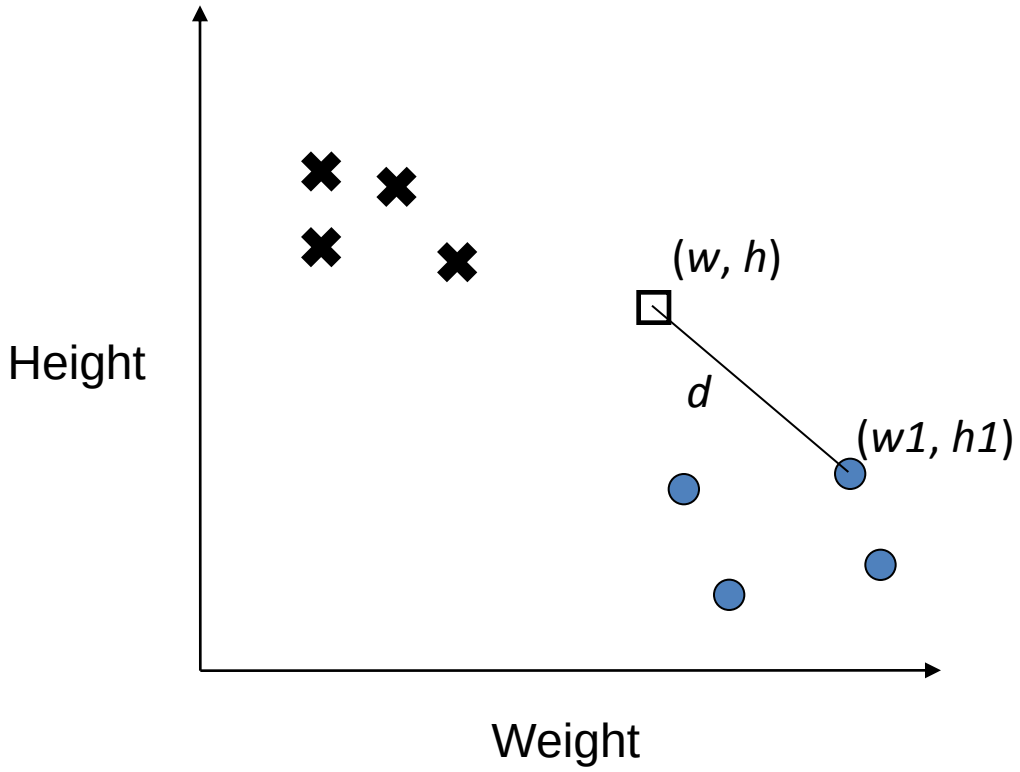
The K-Nearest Neighbour Algorithm

1. *Measure distance to all points*
2. *Find closest "k" points* ← **(here k=3, but it could be more)**
3. *Assign majority class*



“Euclidean distance”

$$d = \sqrt{(w - w_1)^2 + (h - h_1)^2}$$



The K-Nearest Neighbour Algorithm

```
for each testing point
   measure distance to every training point
   find the k closest points
   identify the most common class among those
     k
   predict that class
end
```

- Advantage: Surprisingly good classifier!
- Disadvantage: Have to store the entire training set in memory

Euclidean distance still works in 3-d, 4-d, 5-d, etc.…

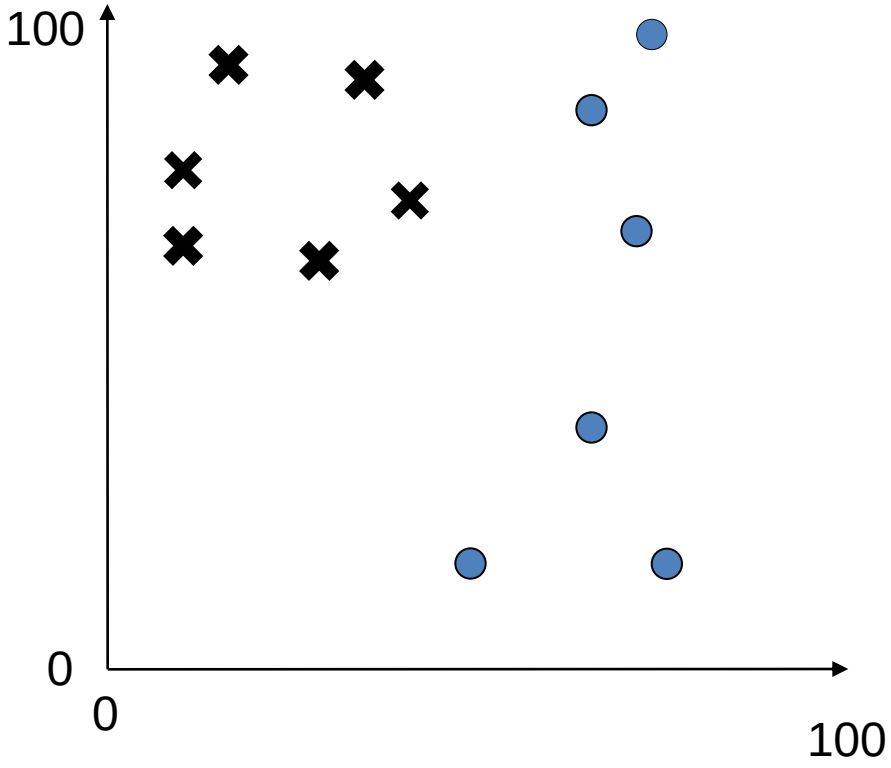
$$d = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2}$$

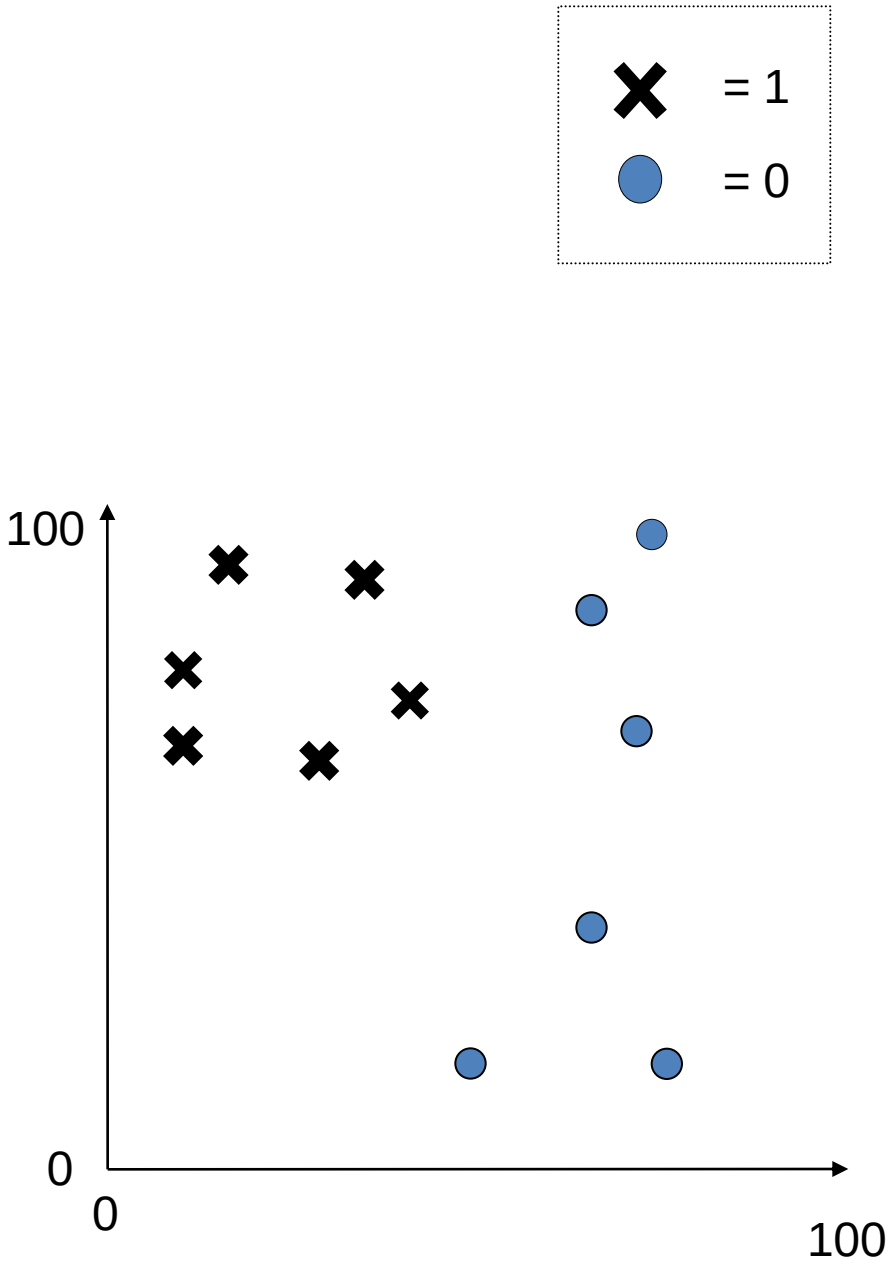
x = Height
y = Weight
z = Shoe size

Choosing the wrong features makes it difficult, too many and it's computationally intensive.

Possible features:
- Shoe size ✓
- Height
- Age ✓
- Weight

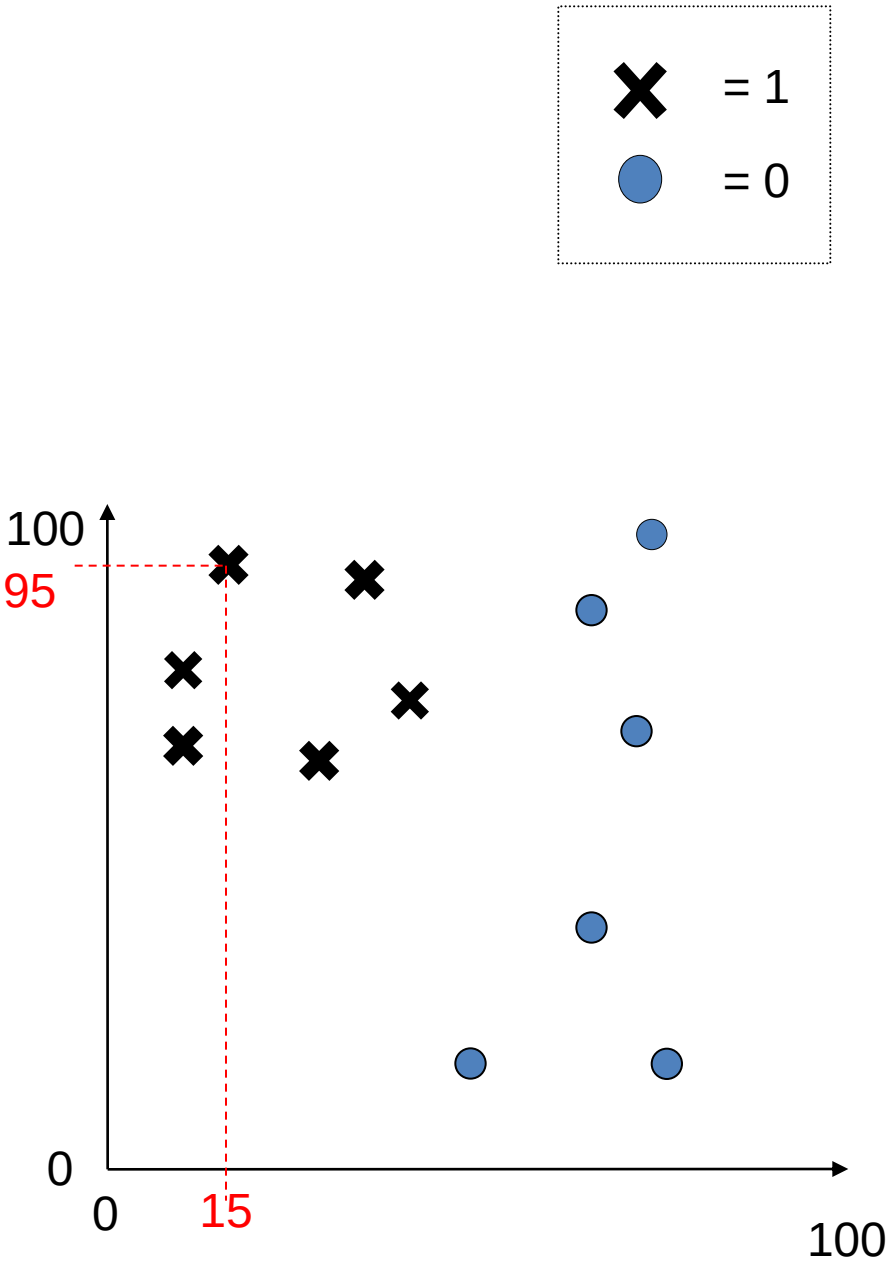
?


100
0
0
100



Inputs

Labels

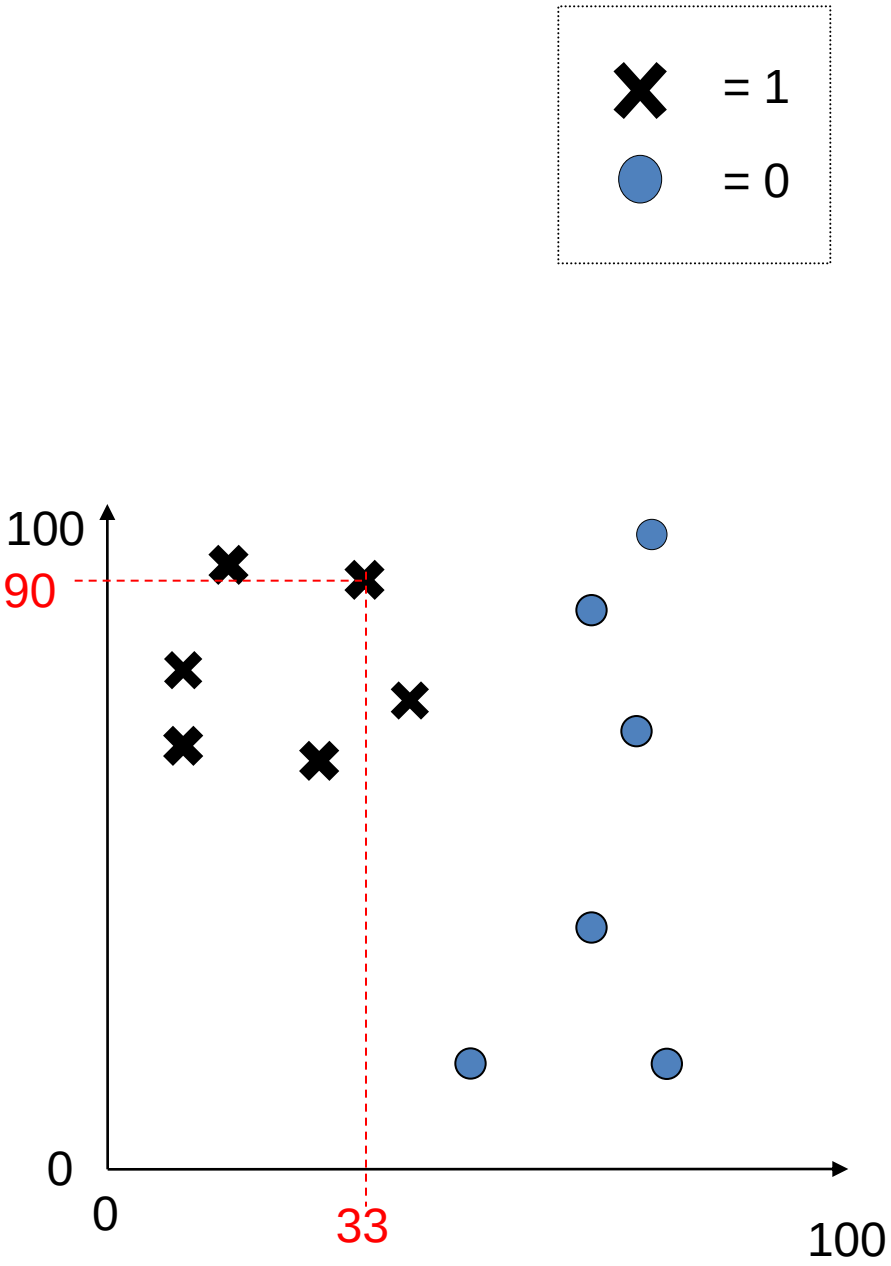


Inputs

15	95

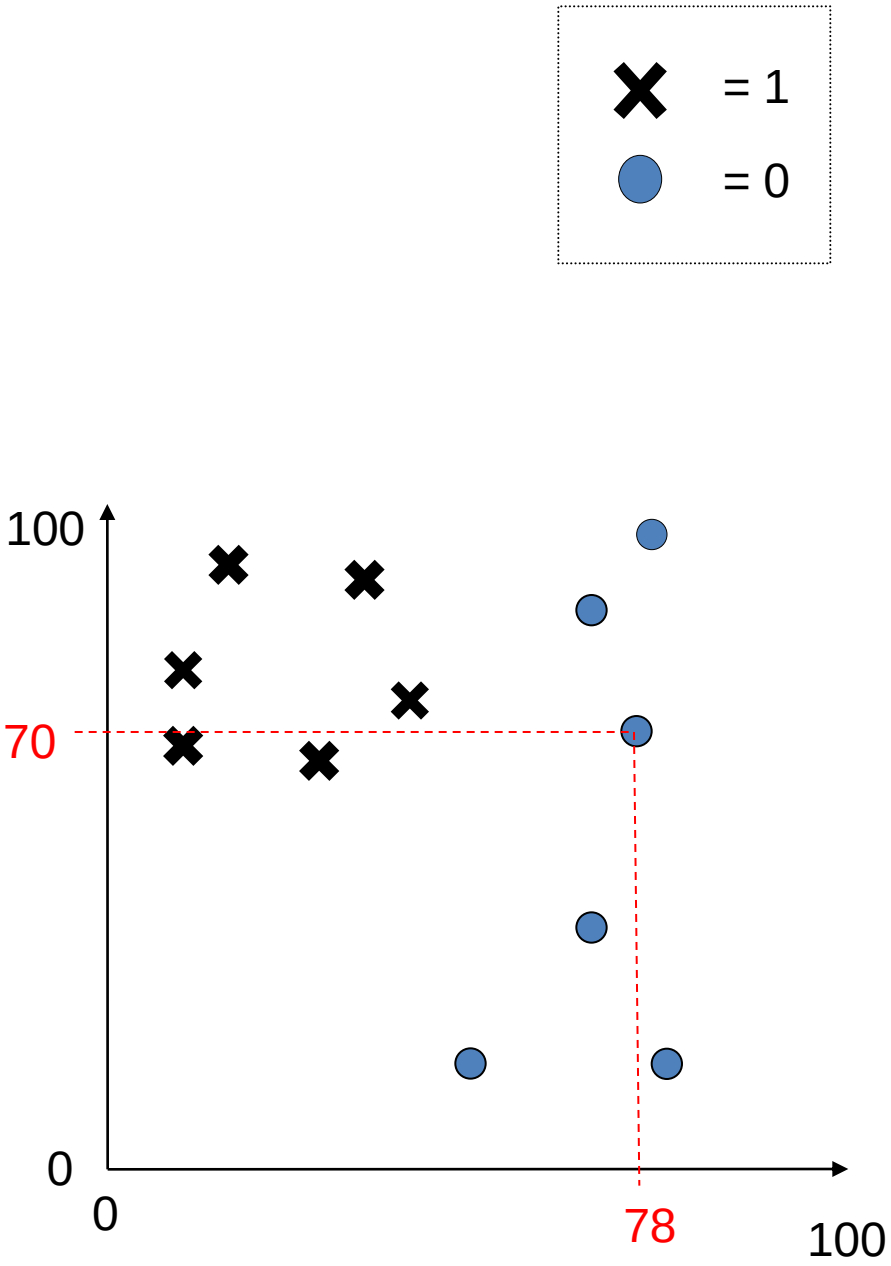
Labels

1



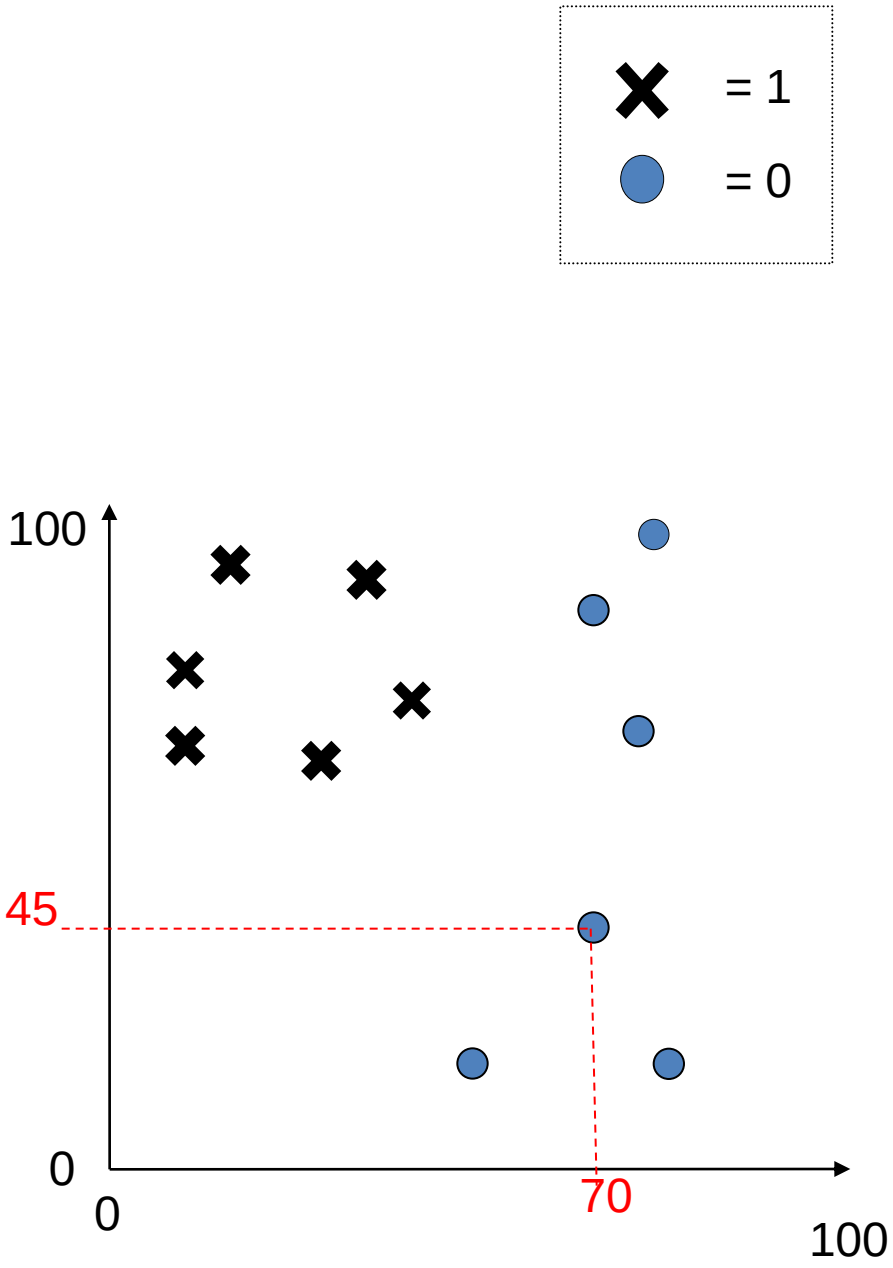
Inputs	
15	95
33	**90**

Labels
1
1



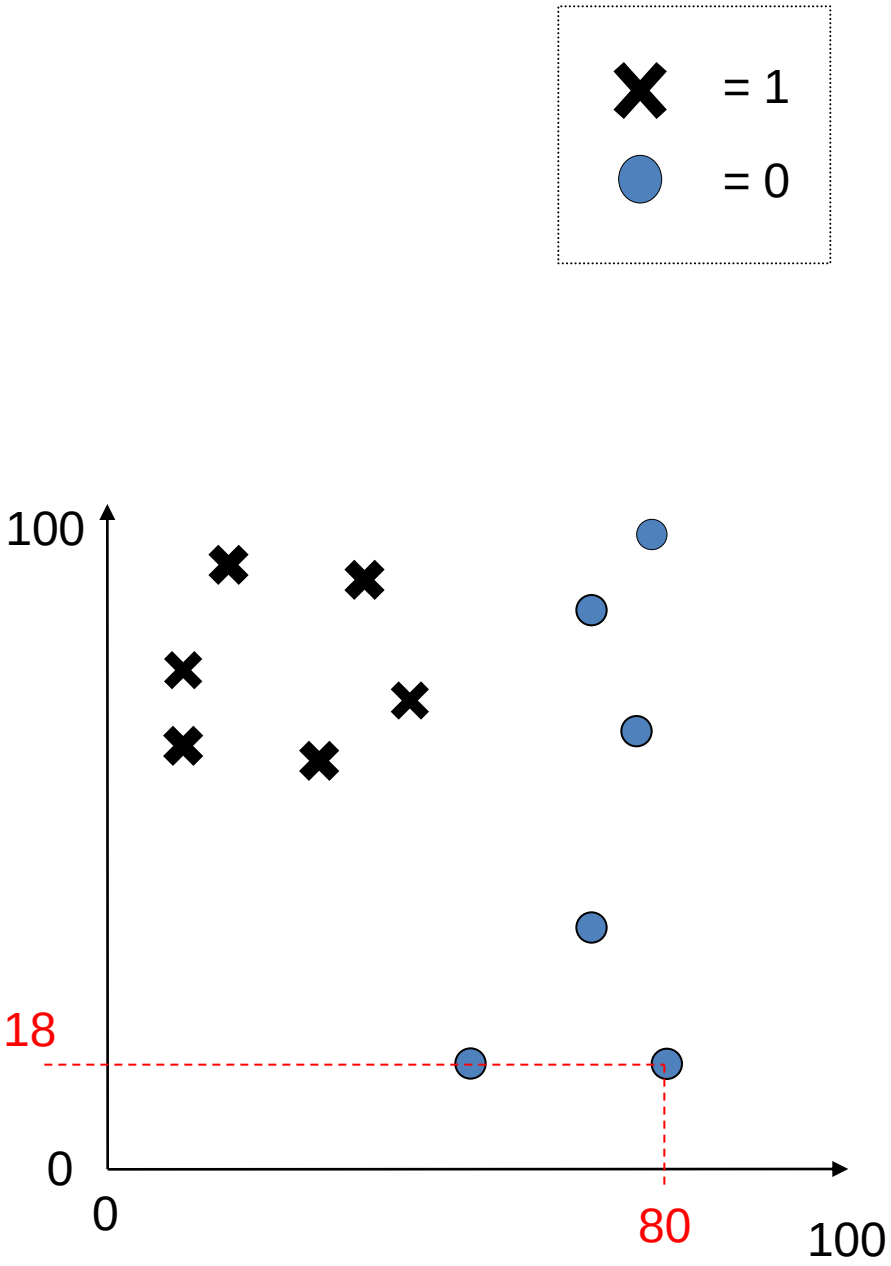
Inputs	
15	95
33	90
78	**70**

Labels
1
1
0



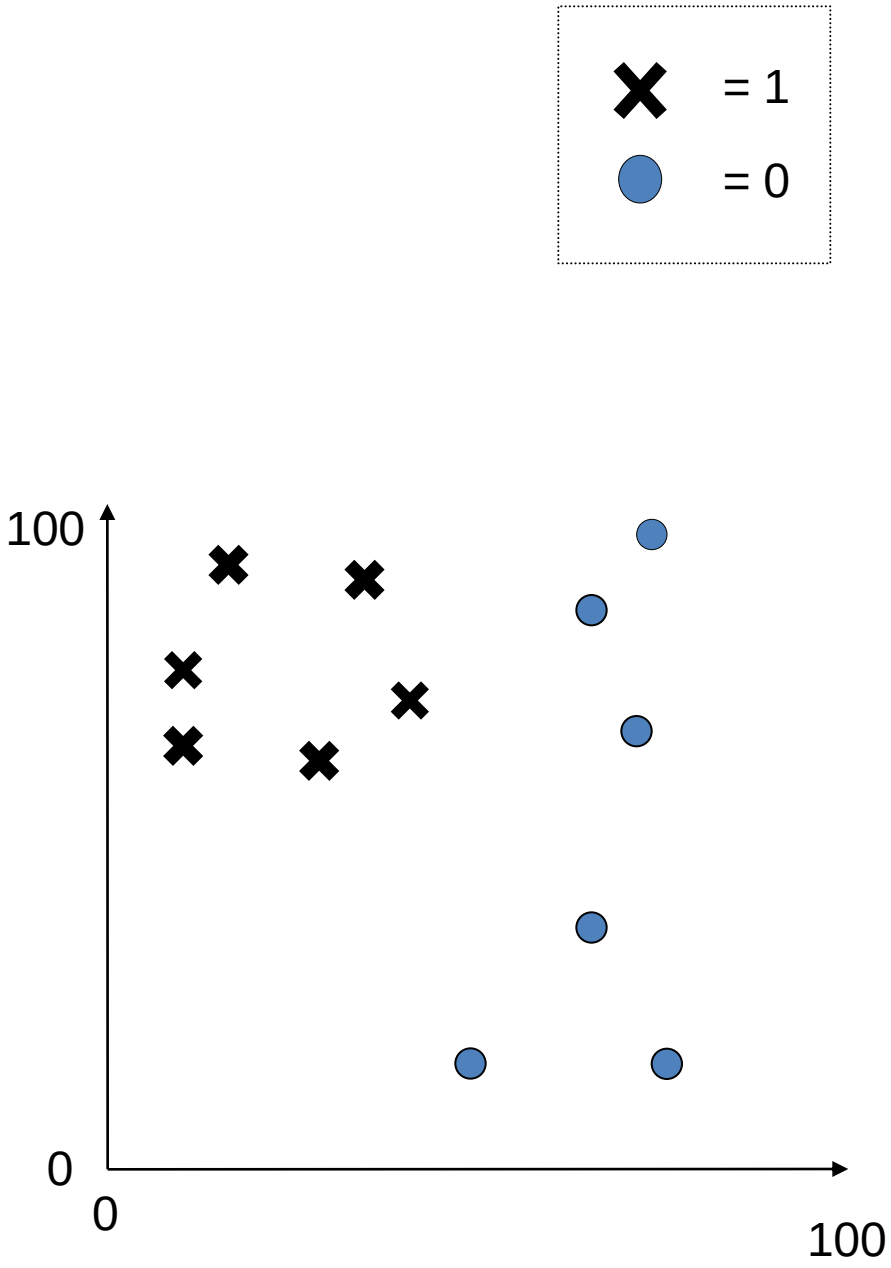
Inputs	
15	95
33	90
78	70
70	**45**

Labels
1
1
0
0



Inputs	
15	95
33	90
78	70
70	45
80	**18**

Labels
1
1
0
0
0

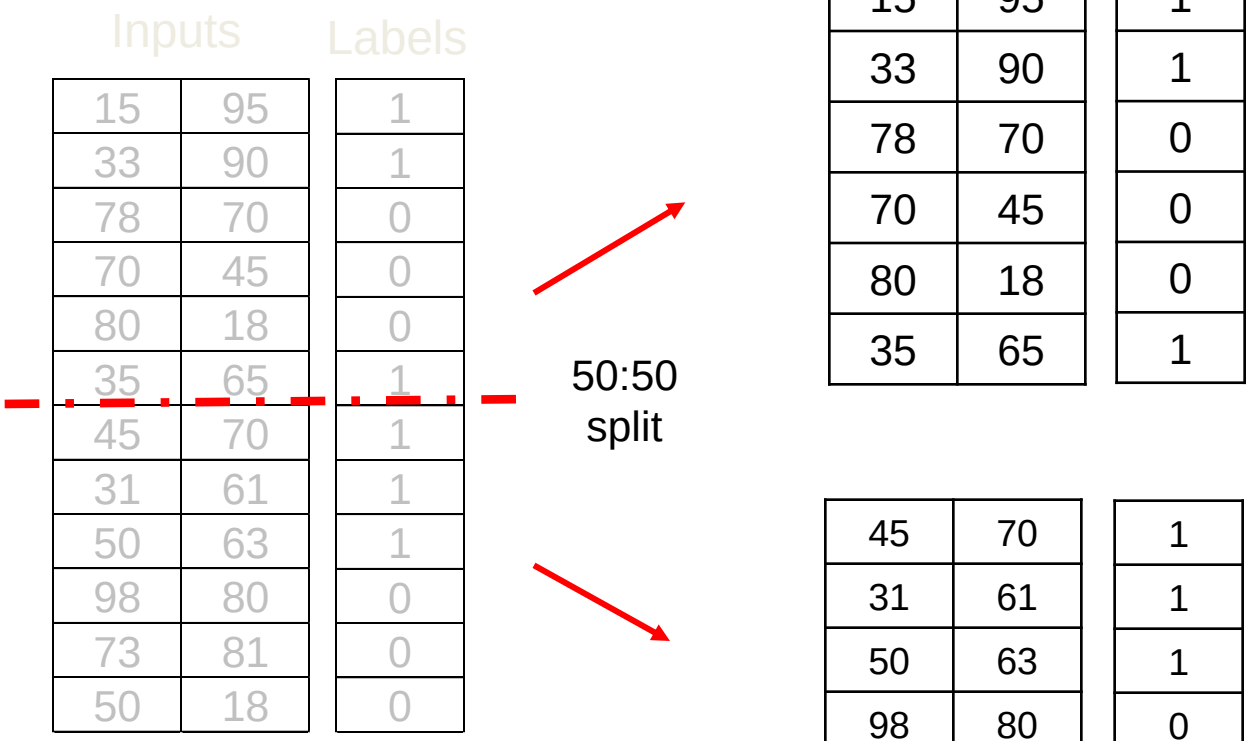


Inputs		Labels	
15	95	1	✖
33	90	1	✖
78	70	0	●
70	45	0	●
80	18	0	●
35	65	1	✖
45	70	1	✖
31	61	1	✖
50	63	1	✖
98	80	0	●
73	81	0	●
50	18	0	●

2 “features”
12 “examples”
2 “classes”

Dataset

Inputs		Labels
15	95	1
33	90	1
78	70	0
70	45	0
80	18	0
35	65	1
45	70	1
31	61	1
50	63	1
98	80	0
73	81	0
50	18	0


Inputs
Labels
15 95 1
33 90 1
78 70 0
70 45 0
80 18 0
35 65 1
45 70 1
31 61 1
50 63 1
98 80 0
73 81 0
50 18 0
50:50
split
15 95 1
33 90 1
78 70 0
70 45 0
80 18 0
35 65 1
45 70 1
31 61 1
50 63 1
98 80 0
73 81 0
50 18 0

Ideally should be small. Smaller is better.

But if too small... you'll make many mistakes on the testing set.

Needs to be quite big. Bigger is better.

15	95	1
33	90	1
78	70	0
70	45	0
80	18	0
35	65	1

Training set

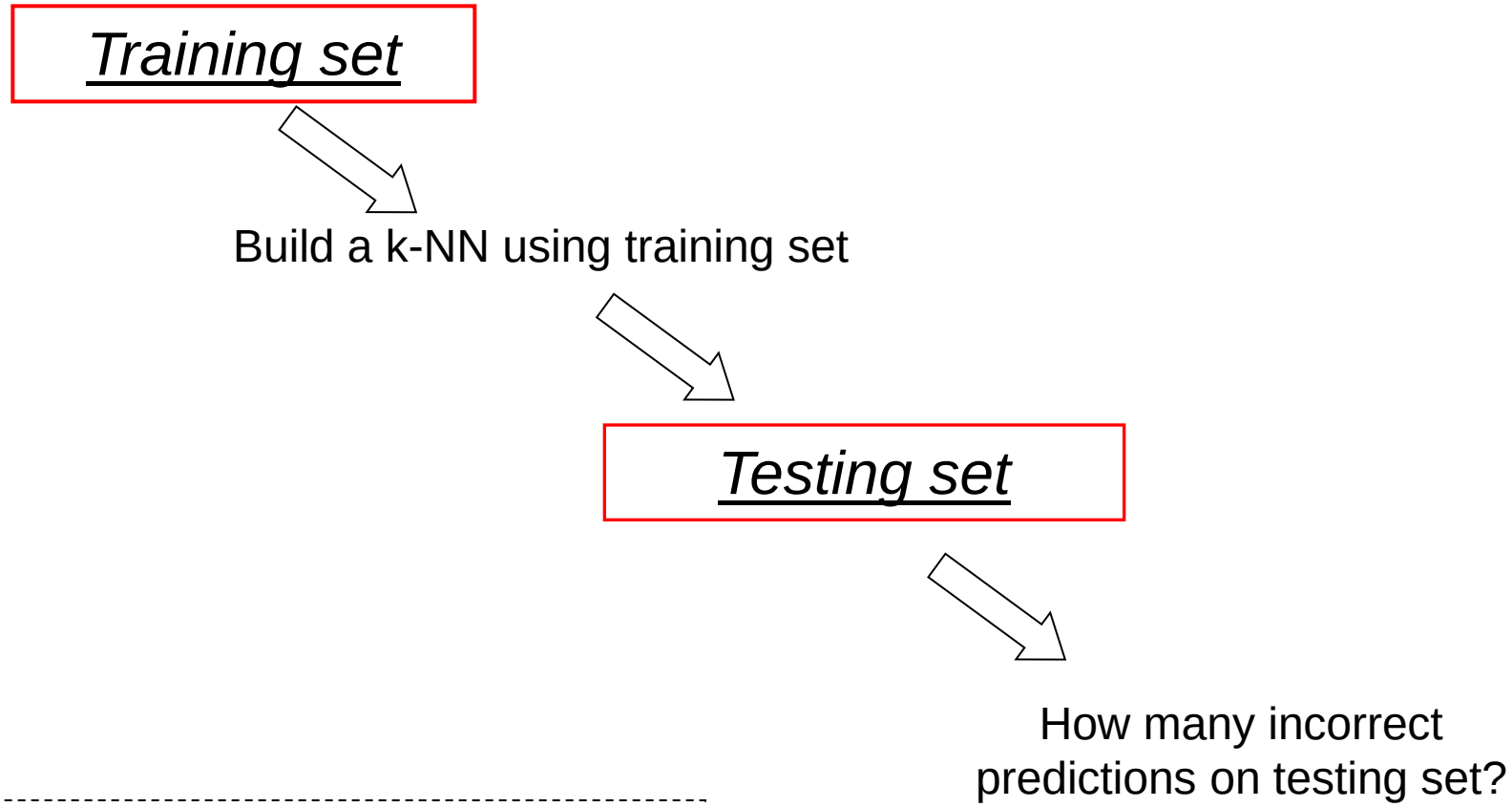
Train a K-NN on this…

45	70	1
31	61	1
50	63	1
98	80	0
73	81	0
50	18	0

Testing set

… then, test it on this!

"simulates" what it might be like to see new data in the future



Percentage of incorrect predictions is called the “error”

e.g. “Training” error
e.g. “Testing” error

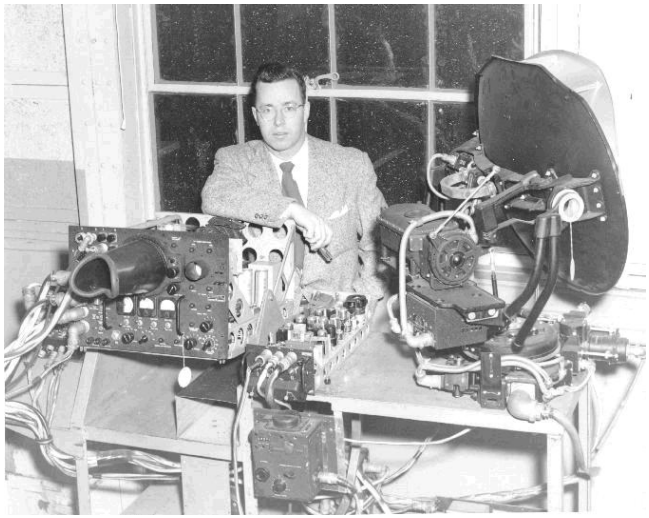
R.O.C. Analysis

32 instances 968 instances

Our obvious solution is in fact backed up a whole statistical framework.

Receiver Operator Characteristics
Developed in WW-2 to assess radar operators.

"How good is the radar operator at spotting incoming bombers?"



False positives
– i.e. falsely predicting a bombing raid

False negatives
– i.e. missing an incoming bomber (VERY BAD!)

R.O.C. Analysis

The “‘3” digits are like the bombers.
Rare events but costly if we misclassify!

False positives – i.e. falsely predicting an event
False negatives – i.e. missing an incoming event

Similarly, we have “true positives” and “true negatives”

		Prediction	
		0	*1*
Truth	*0*	TN	**FP**
	1	**FN**	TP

Minimum Distance
&
Neural Network

Minimum Distance Classifier

- For a test sample X, compute $D_j(X)$ for each class j
- Assign class with minimum D(x) value

$$D_j(\mathbf{x}) = \left\| \mathbf{x} - \mathbf{m}_j \right\|$$

$$= \left[(\mathbf{x} - \mathbf{m}_j)^T (\mathbf{x} - \mathbf{m}_j) \right]^{1/2}$$

- Here m_j is mean value of training samples from j^{th} class

Minimum Distance Classifier

Manipulating $D_j(X)$

$$
\begin{aligned}
D_j^2(\mathbf{x}) &= \left\|\mathbf{x}-\mathbf{m}_j\right\|^2 = (\mathbf{x}-\mathbf{m}_j)^T(\mathbf{x}-\mathbf{m}_j) \\
&= \mathbf{x}^T\mathbf{x}-2\mathbf{x}^T\mathbf{m}_j+\mathbf{m}_j^T\mathbf{m}_j \\
&= \mathbf{x}^T\mathbf{x}-\mathbf{2}\left(\mathbf{x}^T\mathbf{m}_j-\frac{1}{2}\mathbf{m}_j^T\mathbf{m}_j\right).
\end{aligned}
$$

Minimum Distance Classifier

- Now instead of $D_j(X)$, we compute discriminent function $d_j(X)$ for each class
- Assign class with maximum $d_j(X)$ value

$$d_j(\mathbf{x}) = \mathbf{x}^T\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T\mathbf{m}_j \qquad j = 1, 2, \ldots, W$$

- Equation for decision boundary between two classes *i* and *j*

$$d_{ij}(\mathbf{x}) = \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i^T\mathbf{m}_i - \mathbf{m}_j^T\mathbf{m}_j)$$

Minimum Distance Classifier

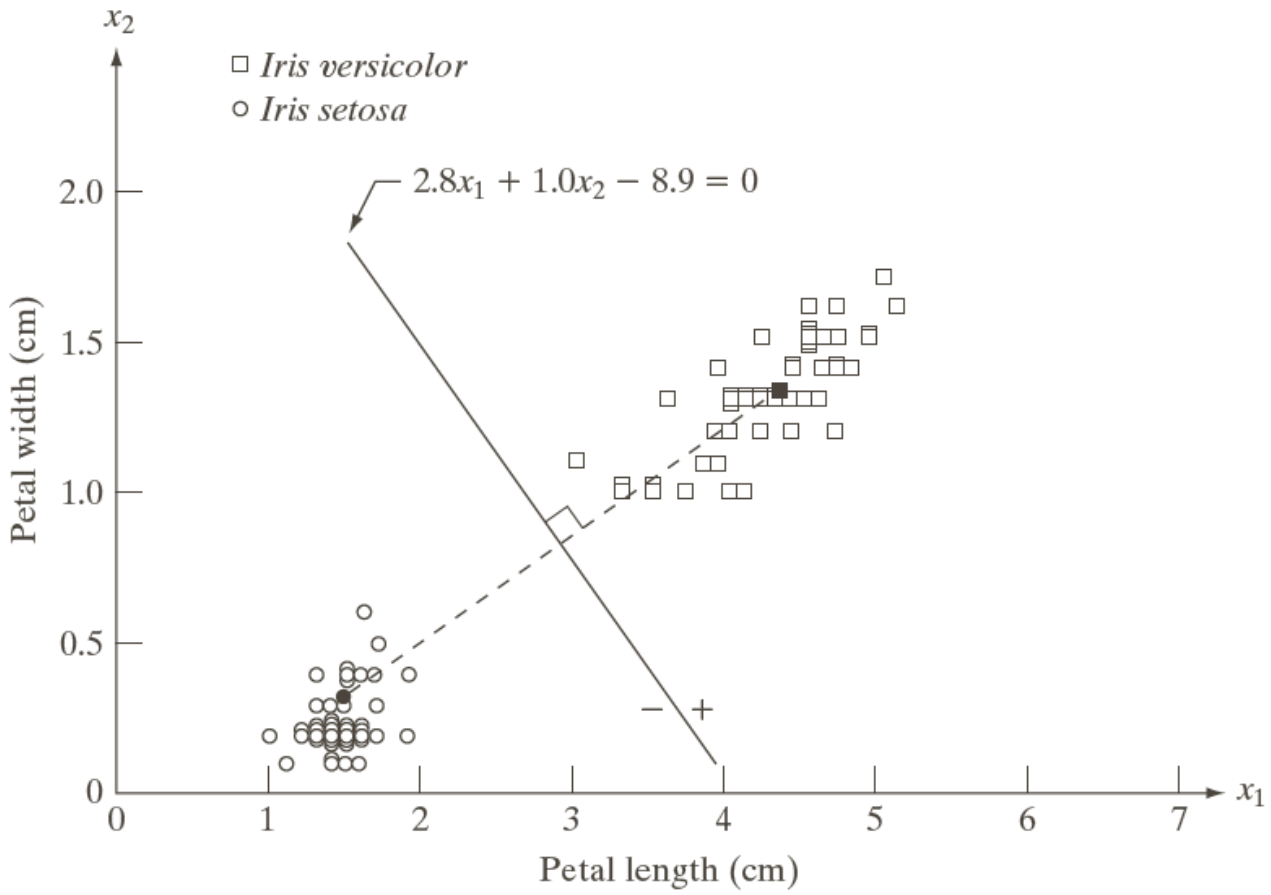
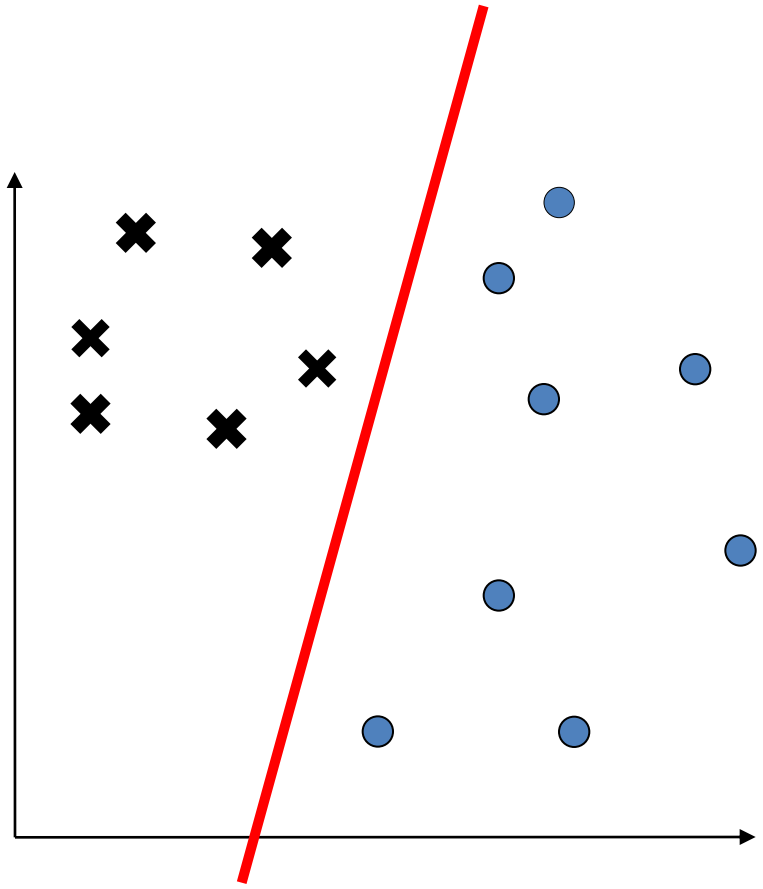


FIGURE 12.6 Decision boundary of minimum distance classifier for the classes of *Iris versicolor* and *Iris setosa*. The dark dot and square are the means.

Artificial Neural Network - Perceptron

A (Linear) Decision Boundary



Represented by:

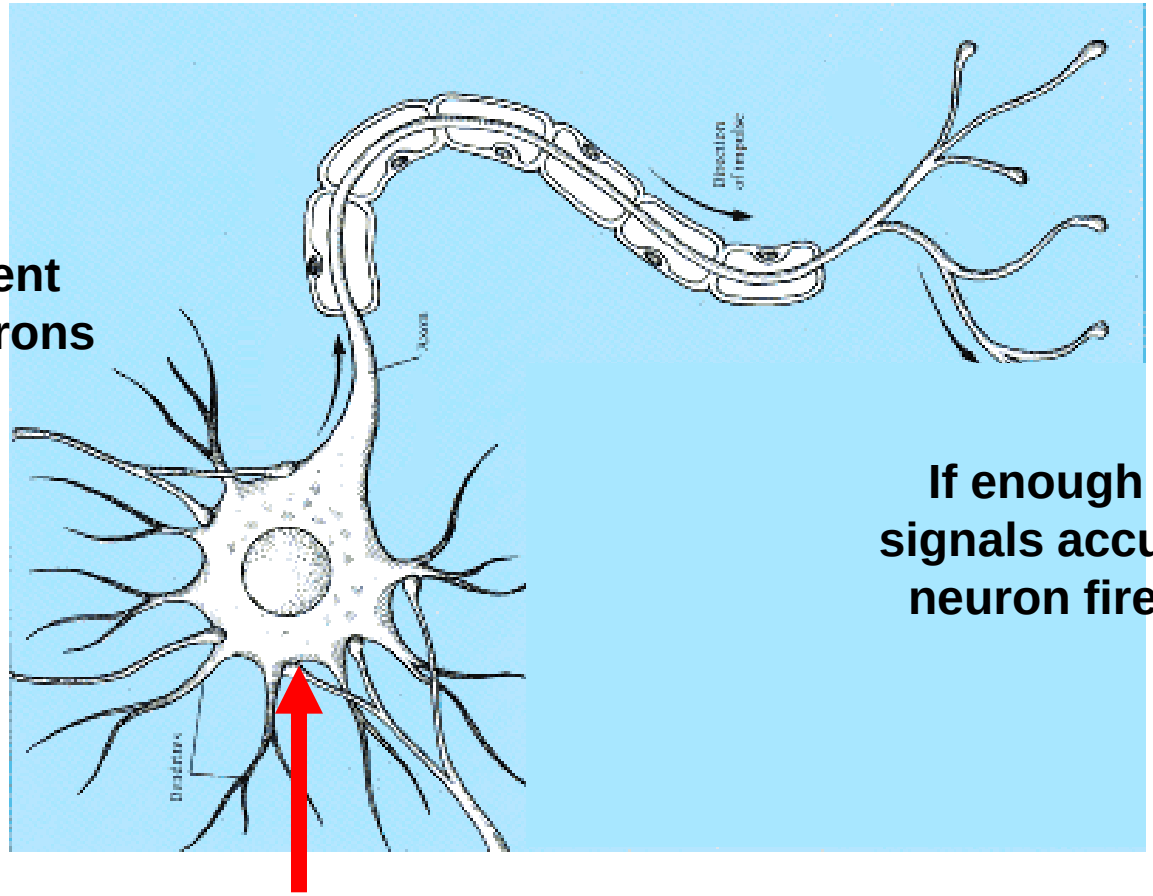
One artificial neuron called a "Perceptron"

Low accuracy (mostly)

Low space complexity

Low time complexity

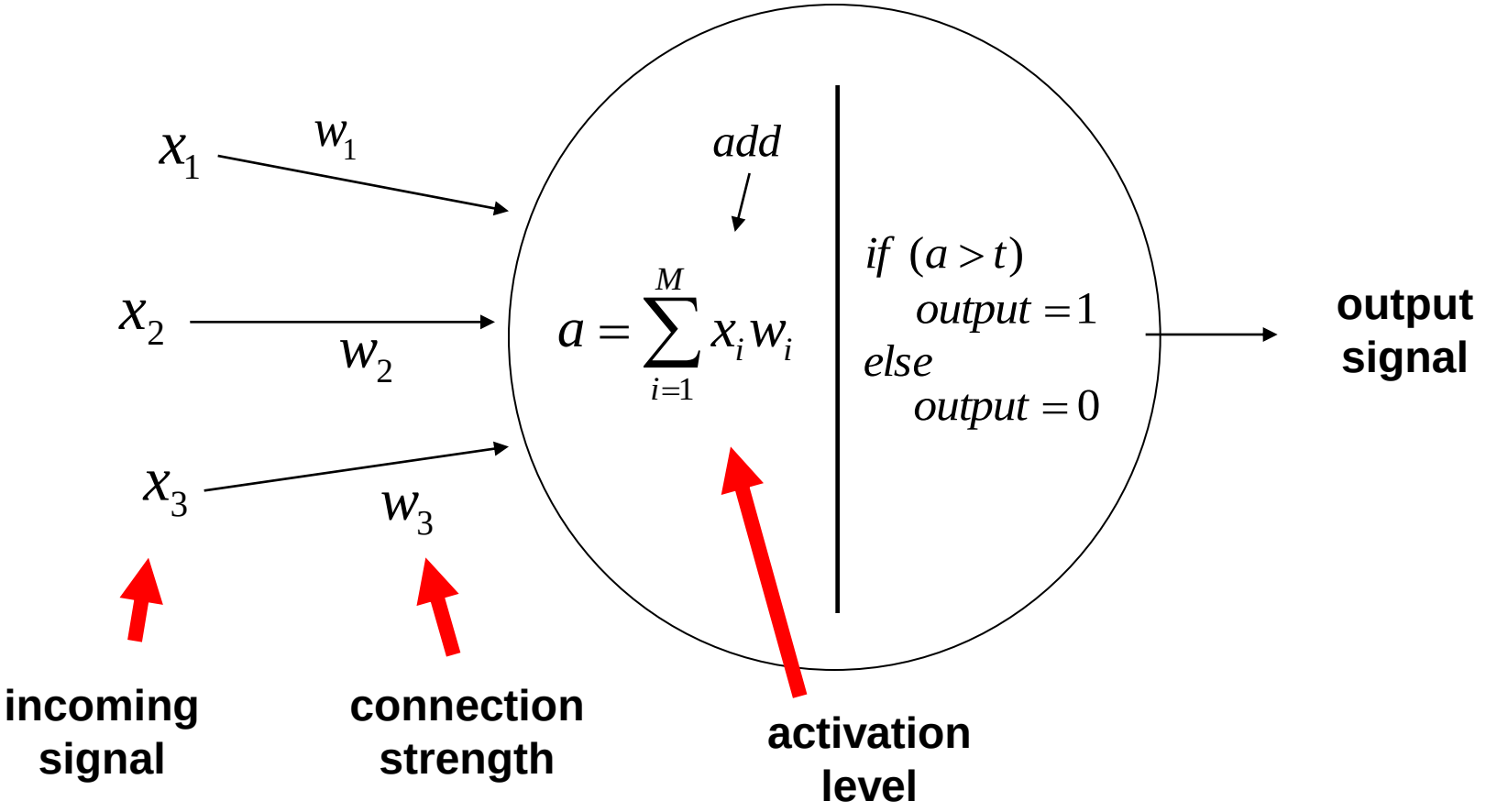
Input signals sent from other neurons



If enough sufficient signals accumulate, the neuron fires a signal.

Connection strengths determine how the signals are accumulated

- input signals ‘x’ and weights ‘w’ are multiplied
- weights correspond to connection strengths
- signals are added up – if they are enough, FIRE!



Calculation…

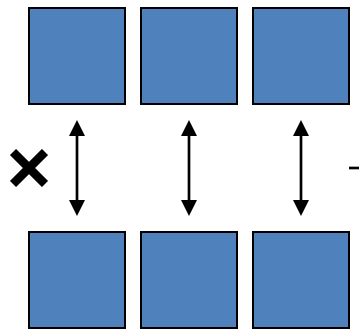
$$a = \sum_{i=1}^{M} x_i w_i$$

Sum notation

(just like a loop from 1 to M)

`double[] x =`

`double[] w =`



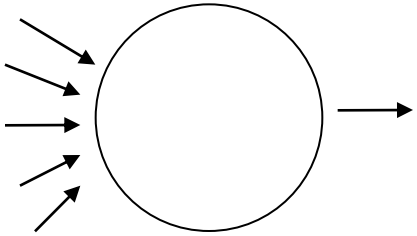
`a` (activation)

Multiple corresponding elements and add them up

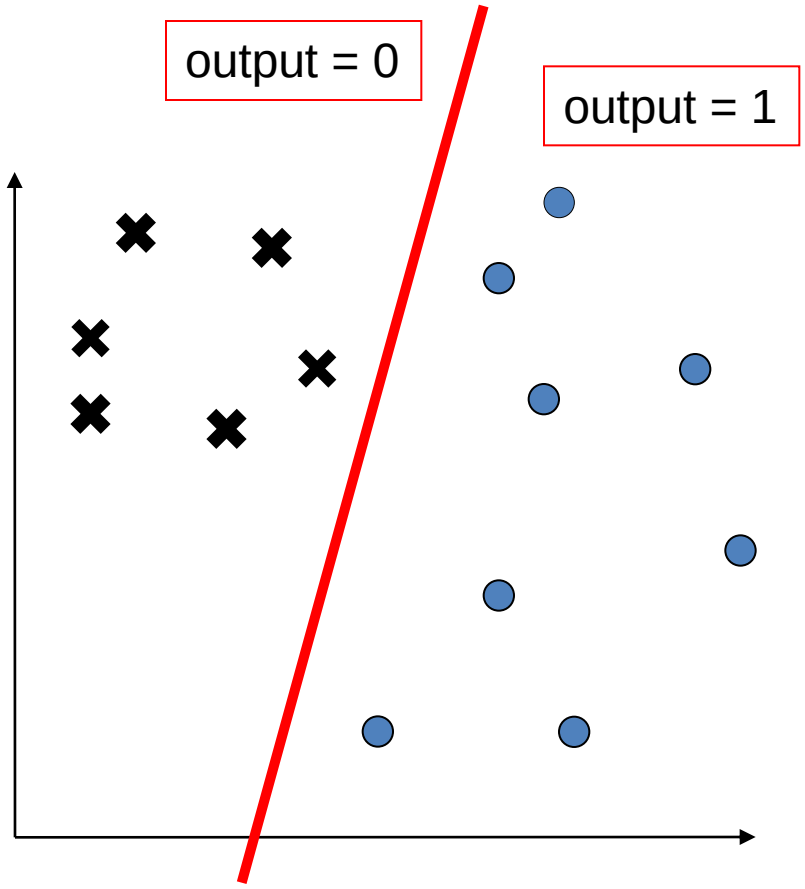
if (activation > threshold) FIRE !

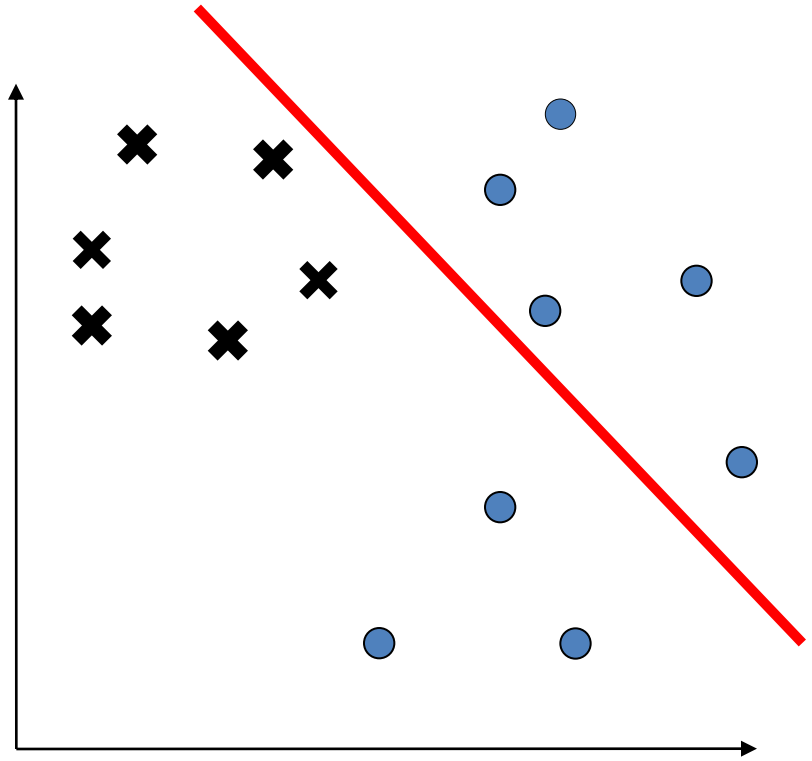
Perceptron Decision Rule

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$



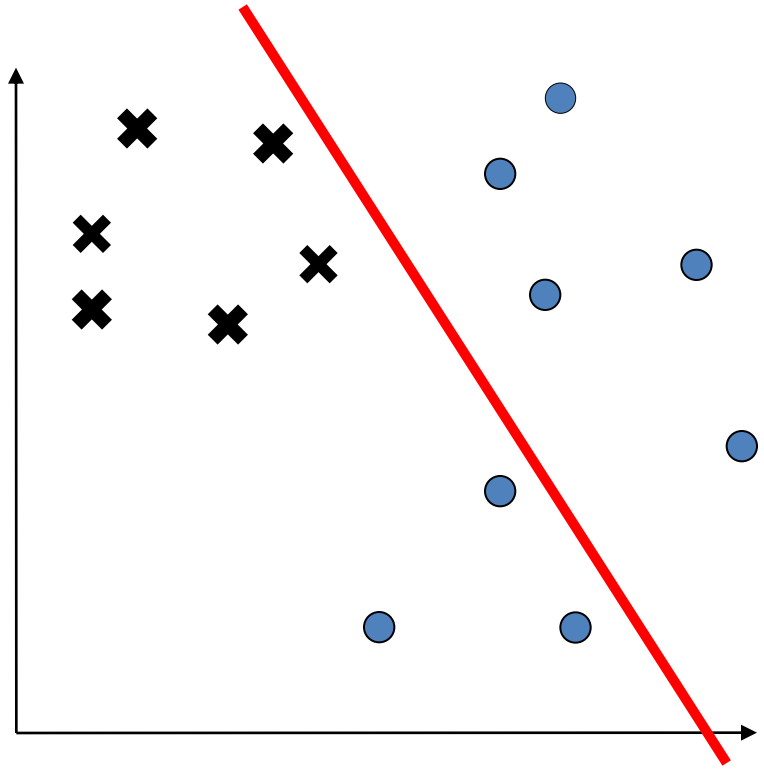
$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$





Is this a good decision boundary?

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

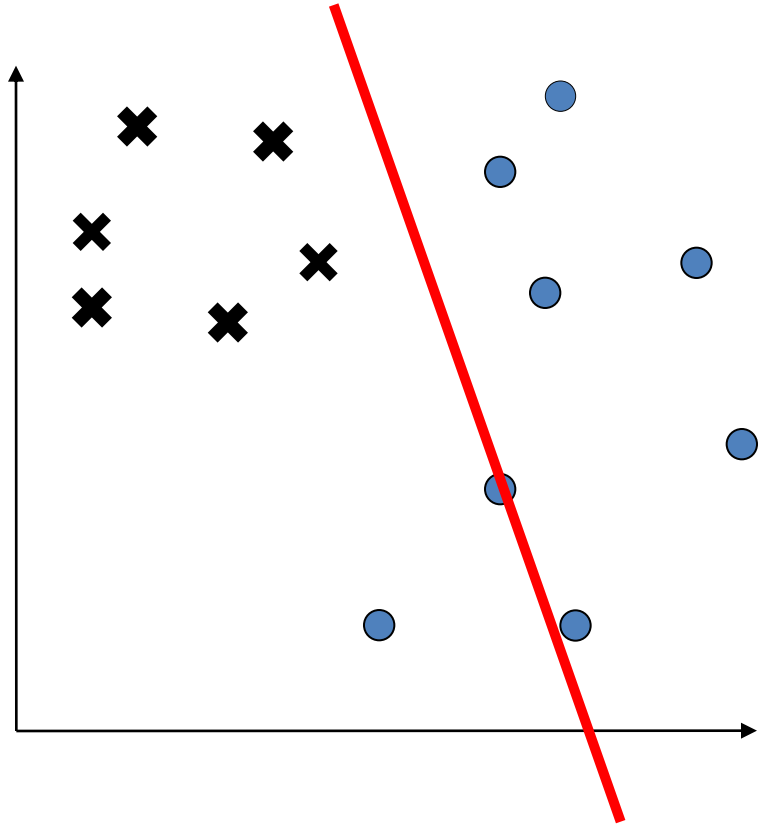


$w_1 = 1.0$

$w_2 = 0.2$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

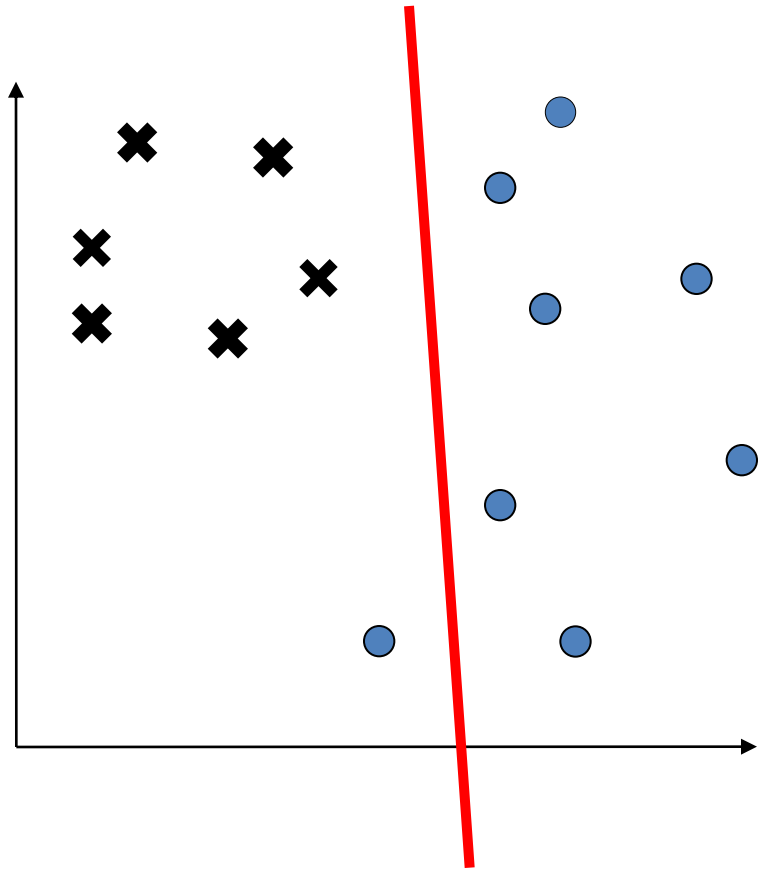


$w_1 = 2.1$

$w_2 = 0.2$

$t = 0.05$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

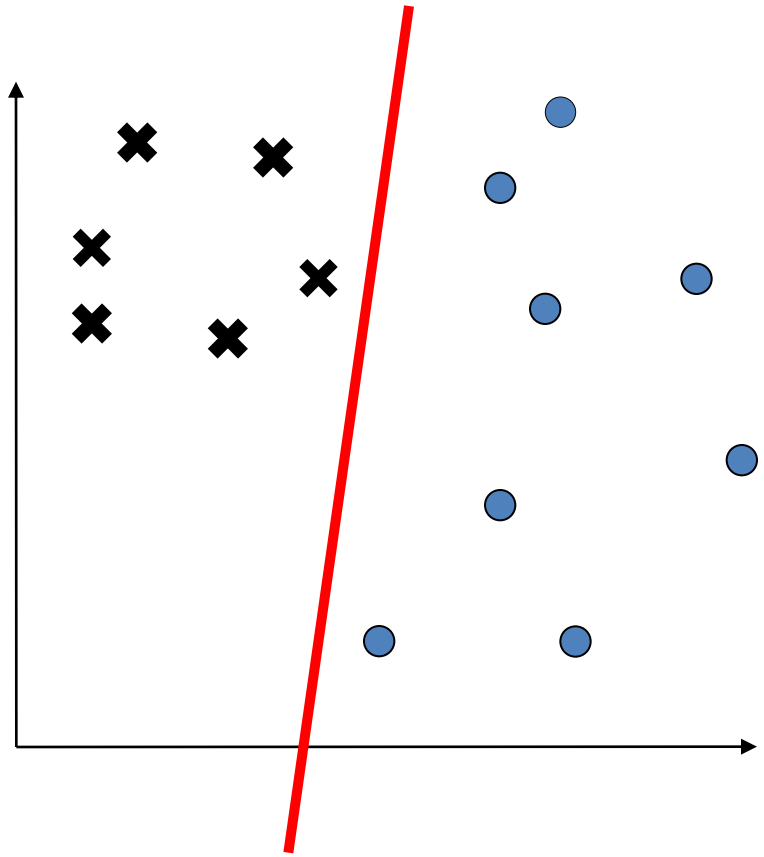


$w_1 = 1.9$

$w_2 = 0.02$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$



$w_1 = -0.8$

$w_2 = 0.03$

$t = 0.05$

Changing the weights/threshold makes the decision boundary move.

Pointless / impossible to do it by hand – only ok for simple 2-D case.

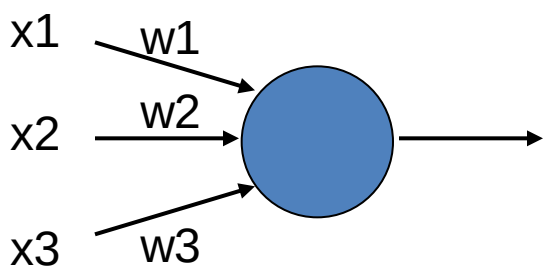
We need an algorithm….

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q1. What is the activation, ***a***, of the neuron?

Q2. Does the neuron fire?

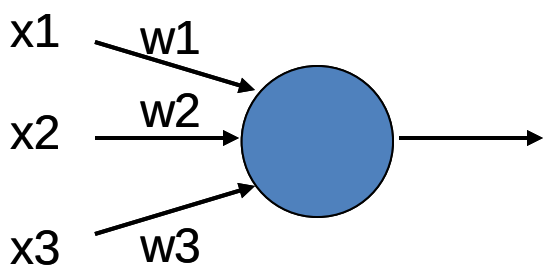
Q3. What if we set threshold at 0.5 and weight #3 to zero?

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q1. What is the activation, *a*, of the neuron?

$$a = \sum_{i=1}^{M} x_i w_i = (1.0 \times 0.2) + (0.5 \times 0.5) + (2.0 \times 0.5) = 1.45$$

Q2. Does the neuron fire?

if (activation > threshold) output=1 else output=0

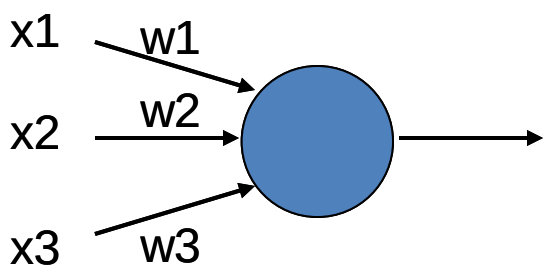
…. So yes, it fires.

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q3. What if we set threshold at 0.5 and weight #3 to zero?

$$a = \sum_{i=1}^{M} x_i w_i = (1.0 \times 0.2) + (0.5 \times 0.5) + (2.0 \times 0.0) = 0.45$$

if (activation > threshold) output=1 else output=0

…. So no, it does not fire..

We can rearrange the decision rule….

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) - t > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) + (-1 \times t) > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) + (x_0 \times w_0) > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=0}^{M} x_i w_i \right) > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

We now treat the threshold like any other weight with a permanent input of -1

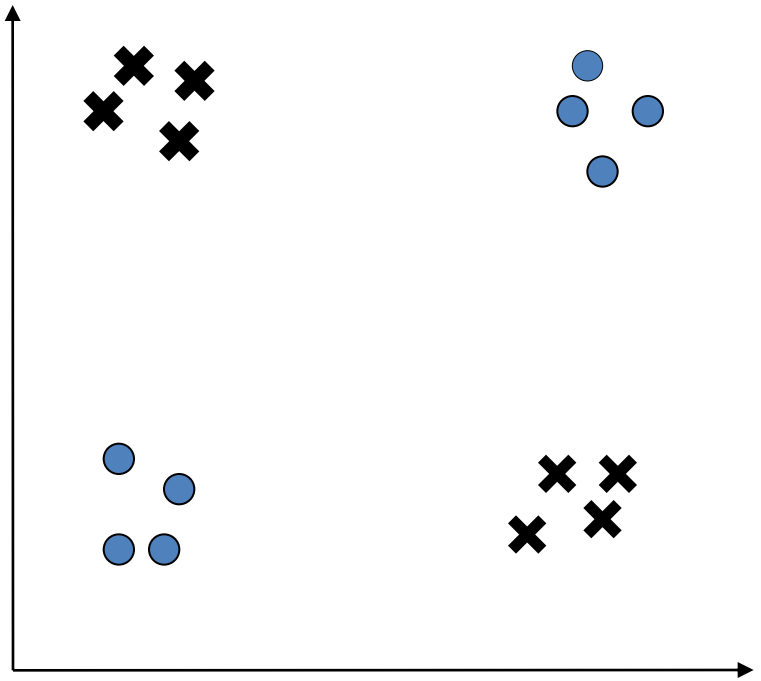
Perceptron Learning Algorithm

```
initialise weights (w)
Repeat until all points are correctly classified
        Repeat for each point
                Calculate margin (yiwXi) for point i)
                If margin > 0, point is correclty classified
                Else change the weights to increase margin such
                that Δw = ηyiXi and wnew = wold + Δw
        end
end
```

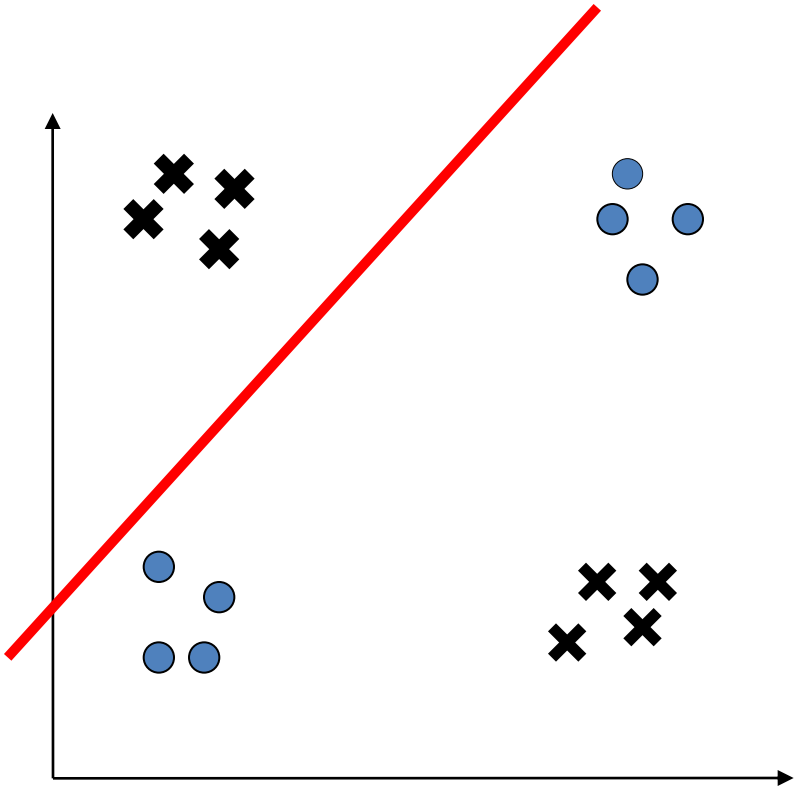
Perceptron convergence theorem:

If the data is linearly separable, then application of the Perceptron learning rule will find a separating decision boundary, within a finite number of iterations

Can a Perceptron solve this problem?



Can a Perceptron solve this problem? ….. NO.



Perceptrons only solve

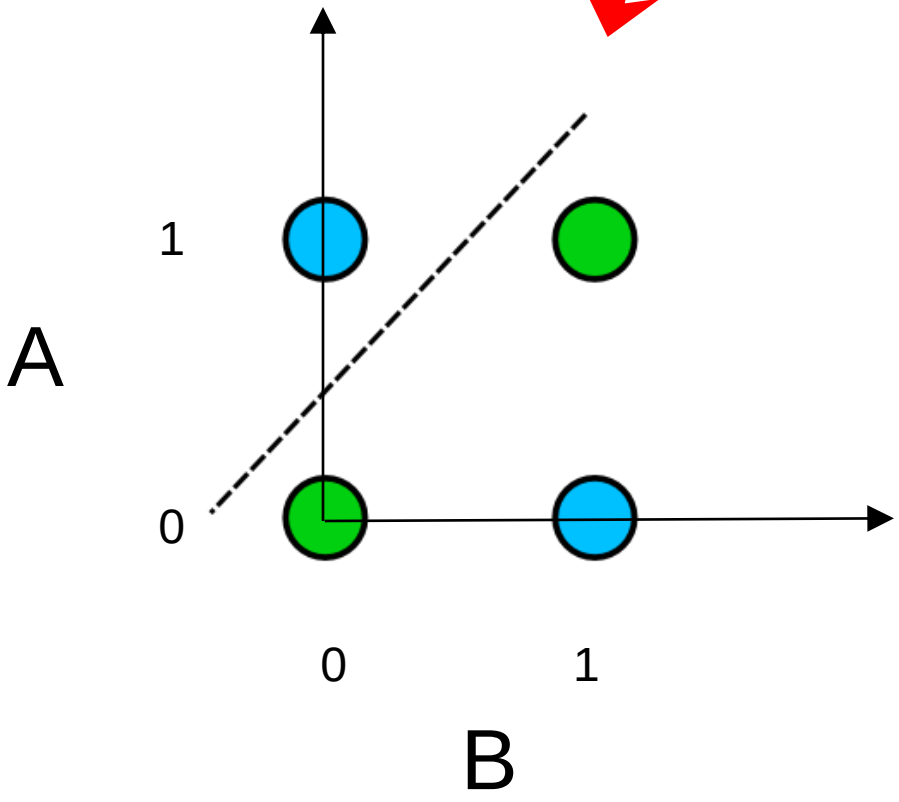
LINEARLY SEPARABLE

problems

With a perceptron…
the decision boundary is
LINEAR

Exclusive-OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



Multilayer Neural Network

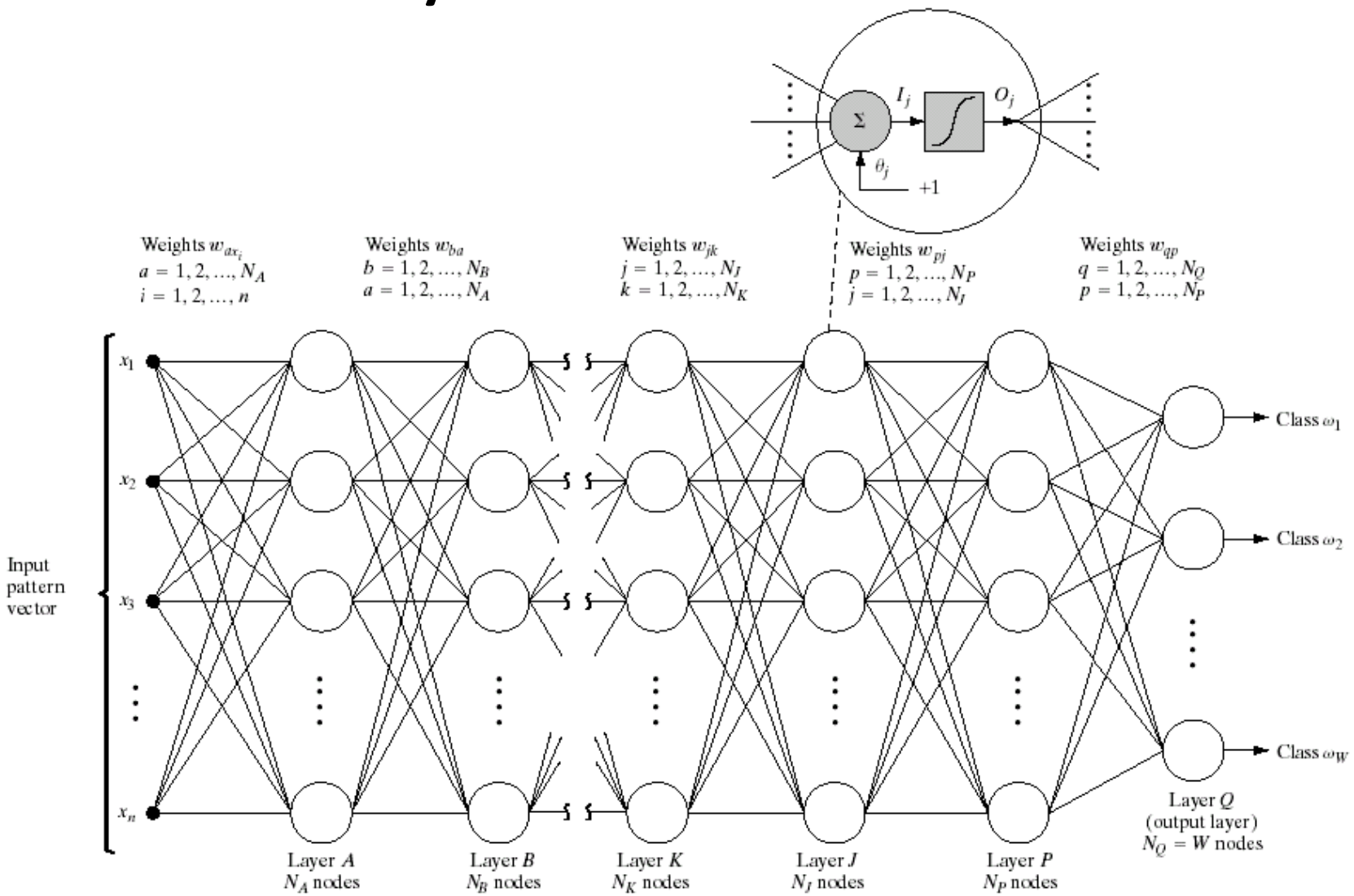


FIGURE 12.16 Multilayer feedforward neural network model. The blowup shows the basic structure of each neuron element throughout the network. The offset, θ_j, is treated as just another weight.

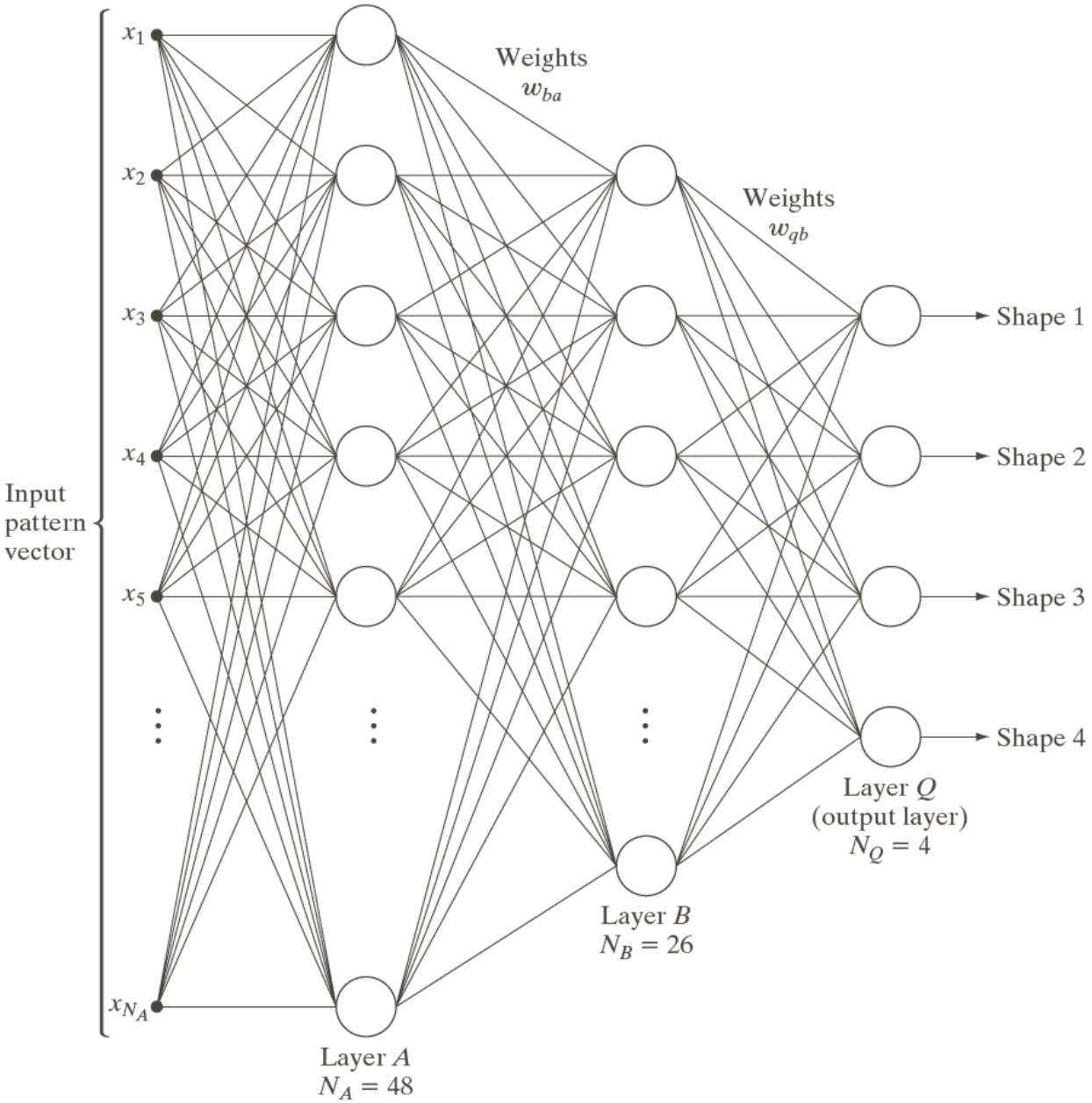
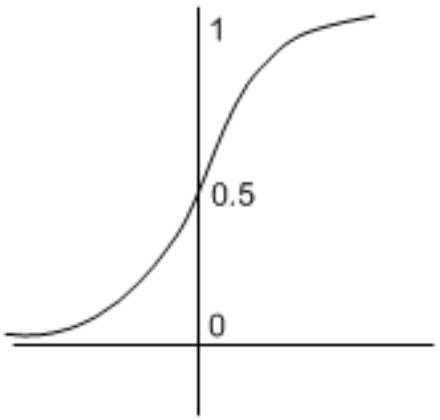


FIGURE 12.19 Three-layer neural network used to recognize the shapes in Fig. 12.18. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

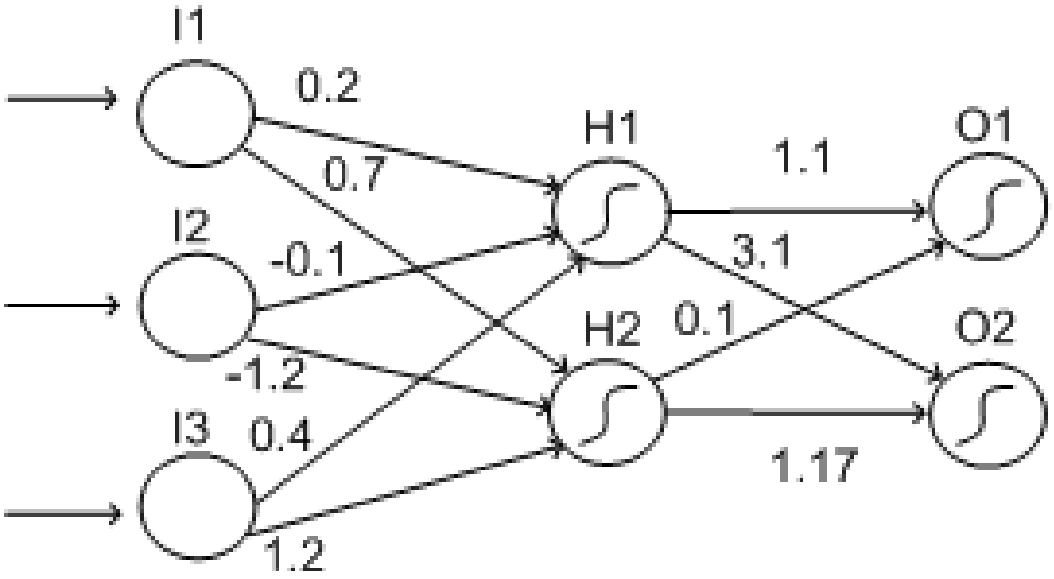
Sigmoid Function

- The function inside units take as input the weighted sum, S, of the values coming from the units connected to it

$$\sigma(S) = \frac{1}{1 + e^{-S}}$$



Example of multilayer Neural Network



- Suppose input values are 10, 30, 20
- The weighted sum coming into H1

$$S_{H1} = (0.2 * 10) + (-0.1 * 30) + (0.4 * 20)$$
$$= 2 -3 + 8 = 7.$$

- The σ function is applied to S_{H1}:

$$\sigma(S_{H1}) = 1/(1+e^{-7}) = 1/(1+0.000912) = 0.999$$

- Similarly, the weighted sum coming into H2:

$$S_{H2} = (0.7 * 10) + (-1.2 * 30) + (1.2 * 20)$$
$$= 7 - 36 + 24 = -5$$

- σ applied to S_{H2}:

$$\sigma(S_{H2}) = 1/(1+e^{5}) = 1/(1+148.4) = 0.0067$$

- Now the weighted sum to output unit O1 :

  $S_{O1} = (1.1 * 0.999) + (0.1*0.0067) = 1.0996$

- The weighted sum to output unit O2:

  $S_{O2} = (3.1 * 0.999) + (1.17*0.0067) = 3.1047$

- The output sigmoid unit in O1:

  $\sigma(S_{O1}) = 1/(1+e^{-1.0996}) = 1/(1+0.333) = 0.750$

- The output from the network for O2:

  $\sigma(S_{O2}) = 1/(1+e^{-3.1047}) = 1/(1+0.045) = 0.957$

- **The input triple (10,30,20) would be categorised with O2, because this has the larger output.**