Artificial Intelligence

Machine Learning: Supervised Learning

Dr Anders Eriksson

What is Machine Learning?

- How to write a computer program that automatically improves its performance through experience
- Machine learning is useful when it is too difficult to come up with a program to perform a desired task
- Make computer to learn by showing examples (most frequently with correct answers)

    - “supervised” learning or learning with a teacher

- In practice: computer program (or function) which has a tunable parameters, tune parameters until the desirable behavior on the examples

Different Types of Learning

- Learning from examples
  - **Unsupervised Learning (previous lecture)**

    Given only inputs as training, find structure in the world: e.g. discover clusters
  - **Supervised Learning (this lecture)**

    Given training examples of inputs and corresponding outputs, produce the correct outputs for new inputs
- Other types, such as **reinforcement learning** will be covered in later lectures

Supervised Machine Learning

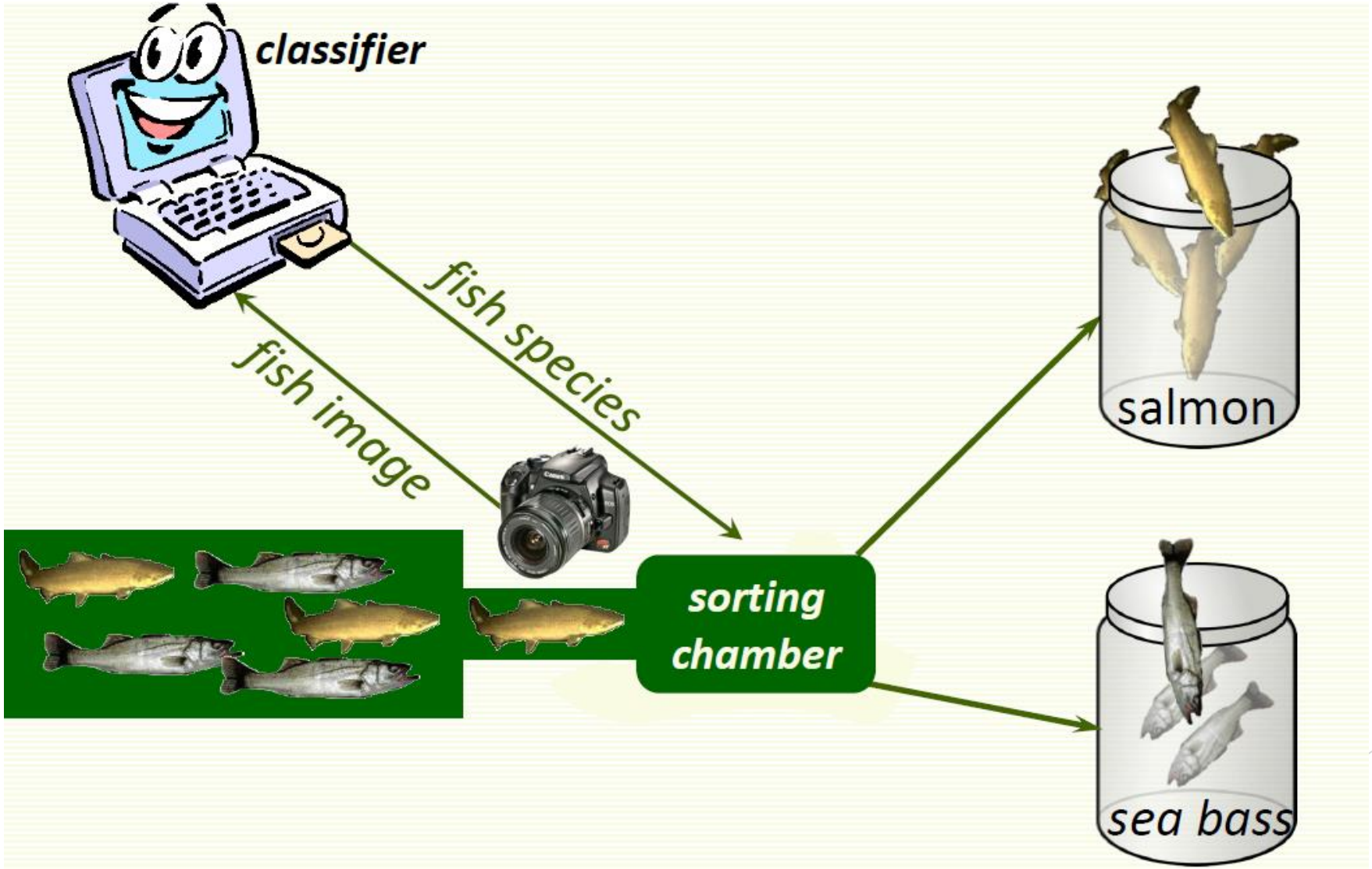
- Training samples (or examples): x^1 , x^2 ,…, x^n
- Each example x^i is typically multi-dimensional
  - x^i_1 , x^i_2 ,…, x^i_d are called features, x^i is often called a feature vector
  - Example: x^1 = [3,7, 35], x^2 = [5, 9, 47], …
  - how many and which features do we take?
- Know desired output for each example y^1 , y^2 ,…, y^n
  - This learning is supervised (“teacher” gives desired outputs)
  - y^i are often one-dimensional
  - Example: y^1 = 1 (“face”), y^2 = 0 (“not a face”)

Two Types of Supervised Machine Learning

- Classification
  - y^i takes value in finite set, called a label or a class
  - Example: y^i ={“sunny”, ”cloudy”, ”raining”}

- Regression
  - y^i continuous, called an output value
  - Example: y^i = temperature [-60,60]

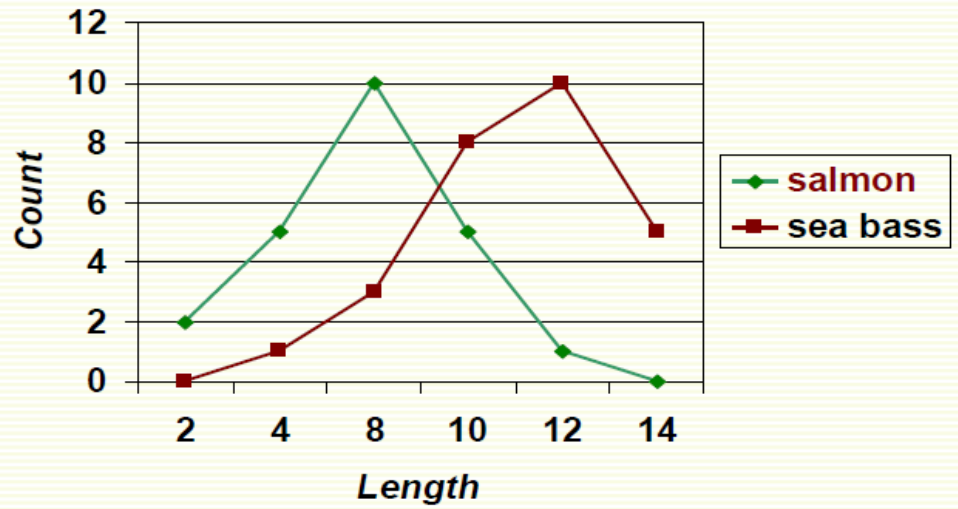
Example : Fish sorting



Example : Fish sorting – Classifier design

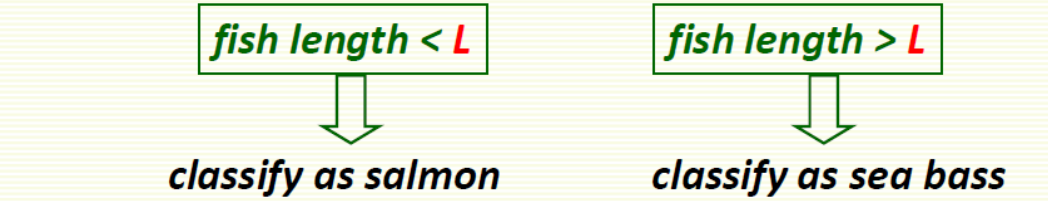
- Notice salmon tends to be shorter than sea bass
- Use ***fish length*** as the discriminating feature
- Count number of bass and salmon of each length

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0



Example : Fish sorting - Length Classifier

- Find the best length L threshold

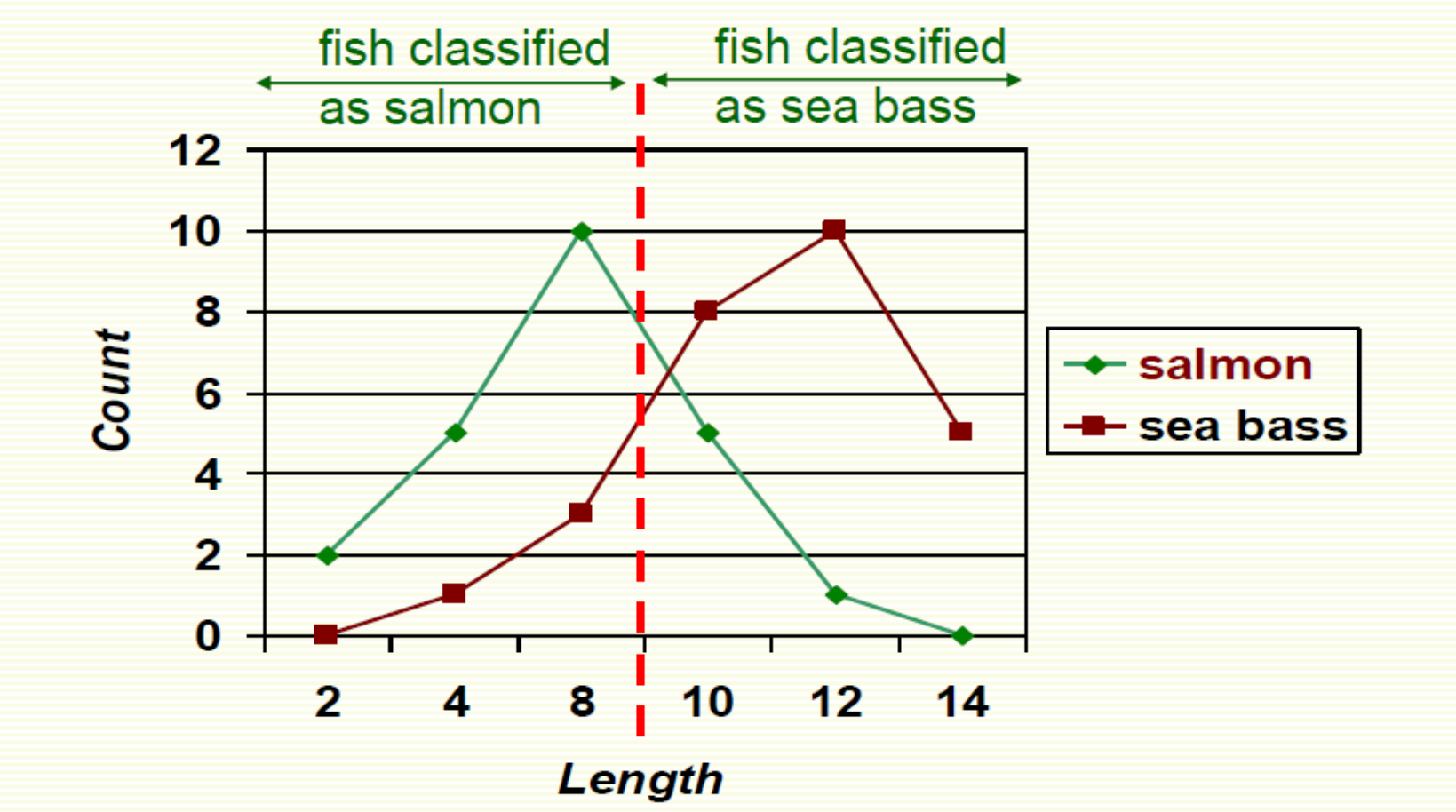


- For example, at ***L*** = 5, misclassified:
  - 1 sea bass
  - 16 salmon

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0

- Classification error (total error) ***17/50 = 34%***

Example : Fish sorting - Length Classifier

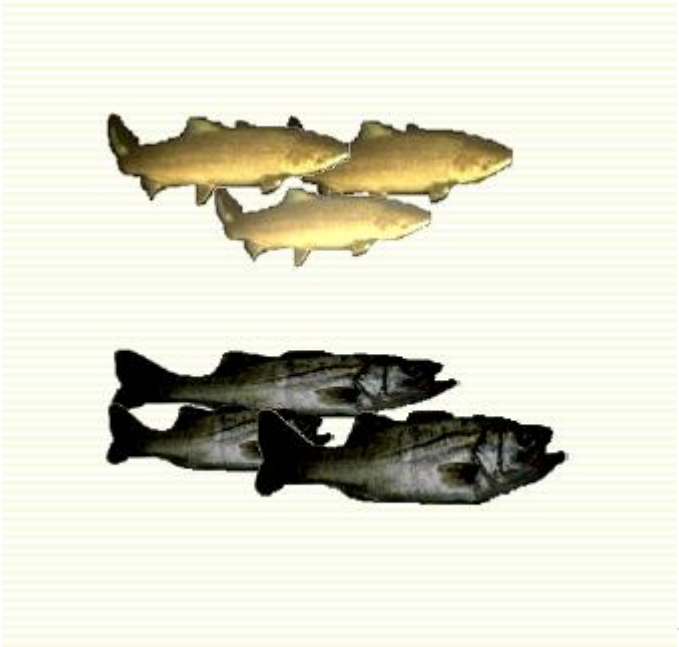


- After searching through all possible thresholds *L*, the best *L*= 9, and still 20% of fish is misclassified

Example : Fish sorting – Lesson Learned?

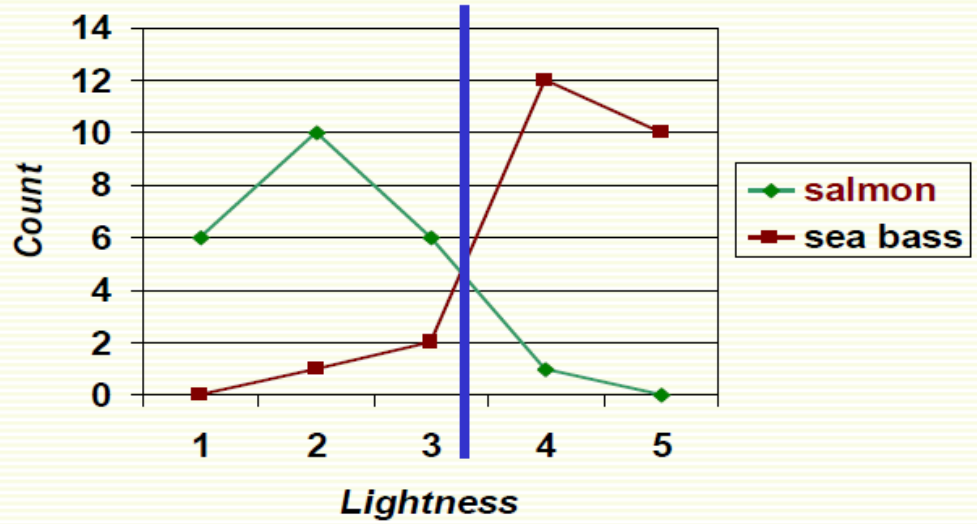
- Length is a poor feature alone!

- What to do?
- Try another feature
- Salmon tends to be lighter
- Try average fish lightness



Example : Fish sorting – Lightness classifier

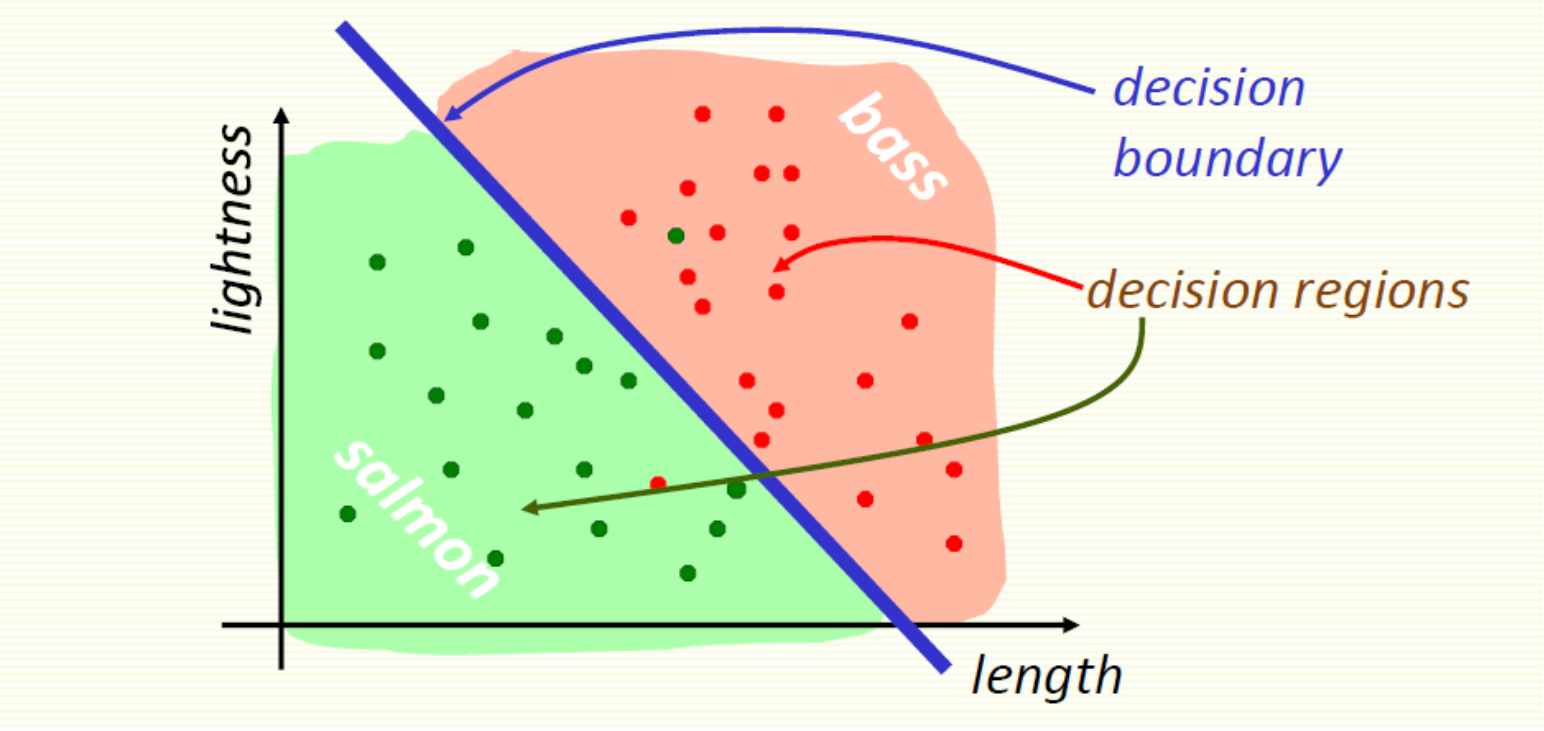
	1	2	3	4	5
bass	0	1	2	10	12
salmon	6	10	6	1	0



- Now fish are classified best at lightness threshold of **L=3.5** with classification error of **8%**

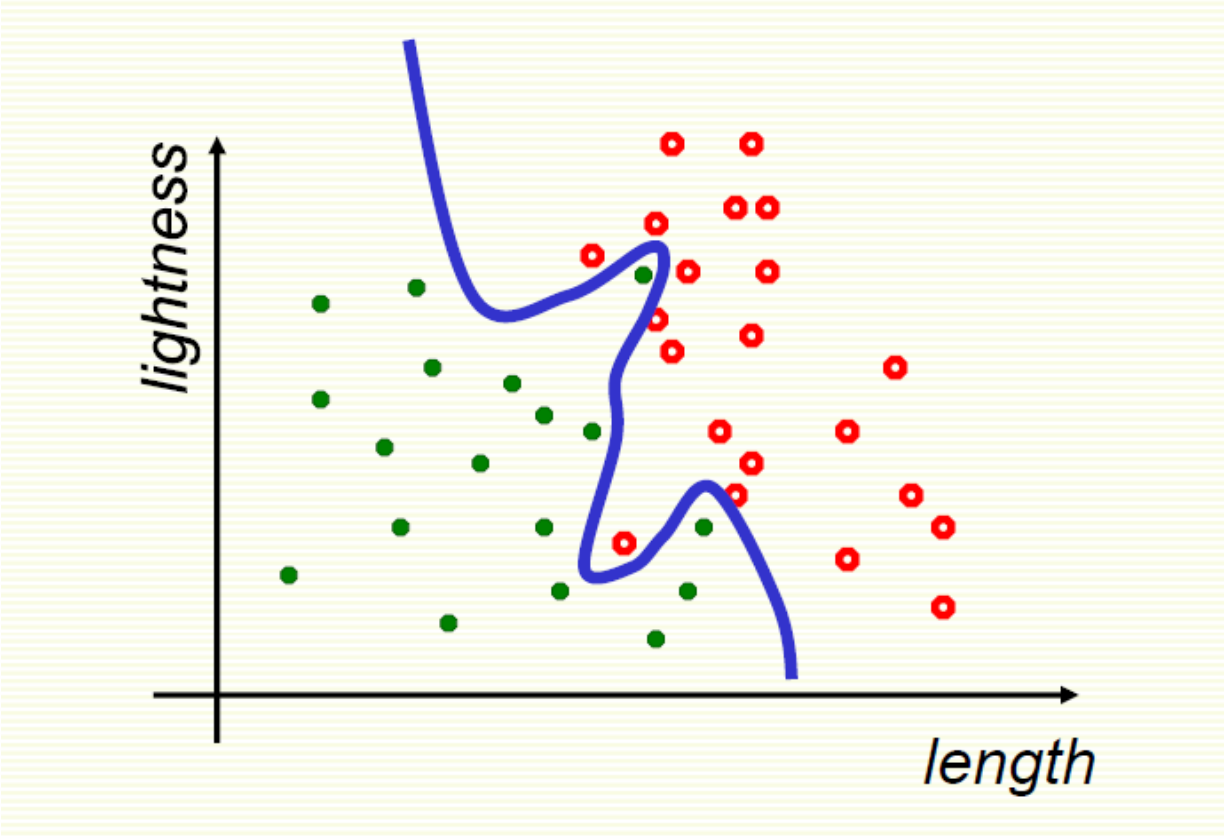
Example : Fish sorting – Combining Features

- Use both length and lightness features
- Feature vector [length,lightness]



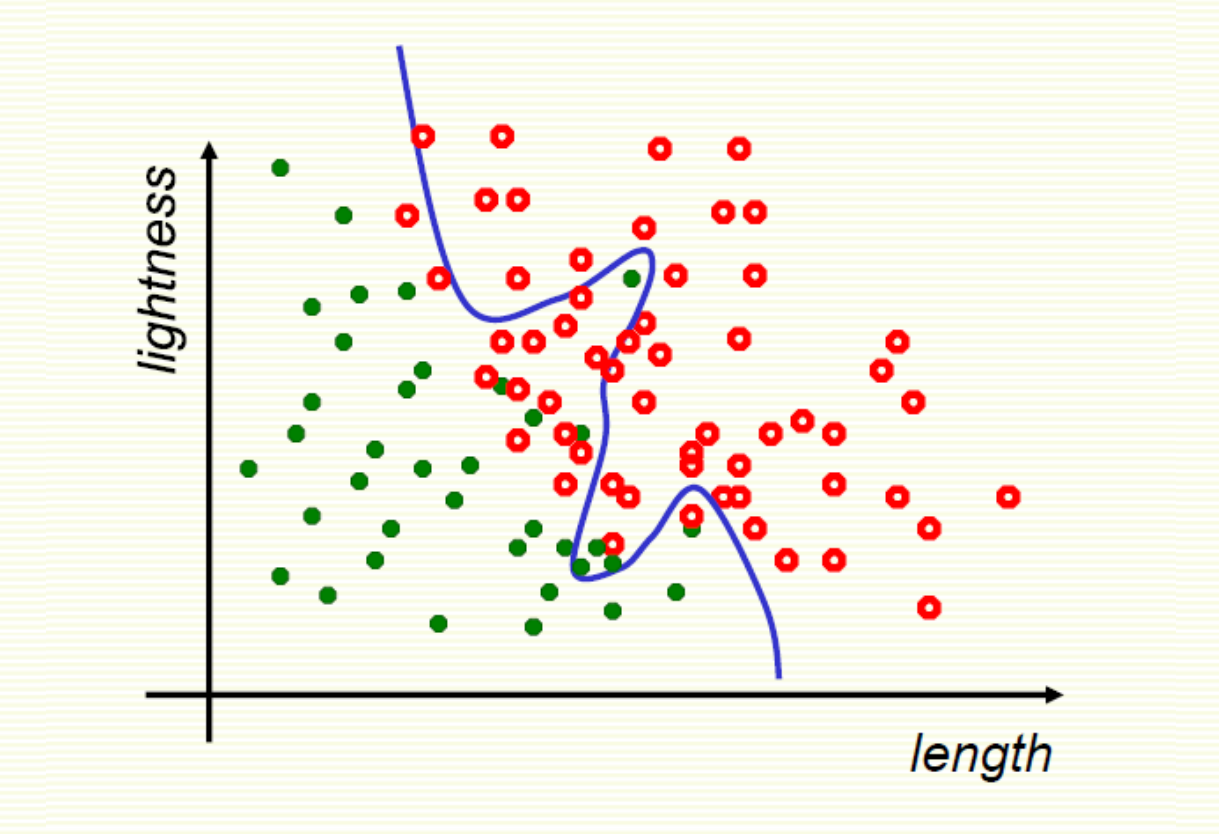
- Classification error 4%

Example : Fish sorting – Even better



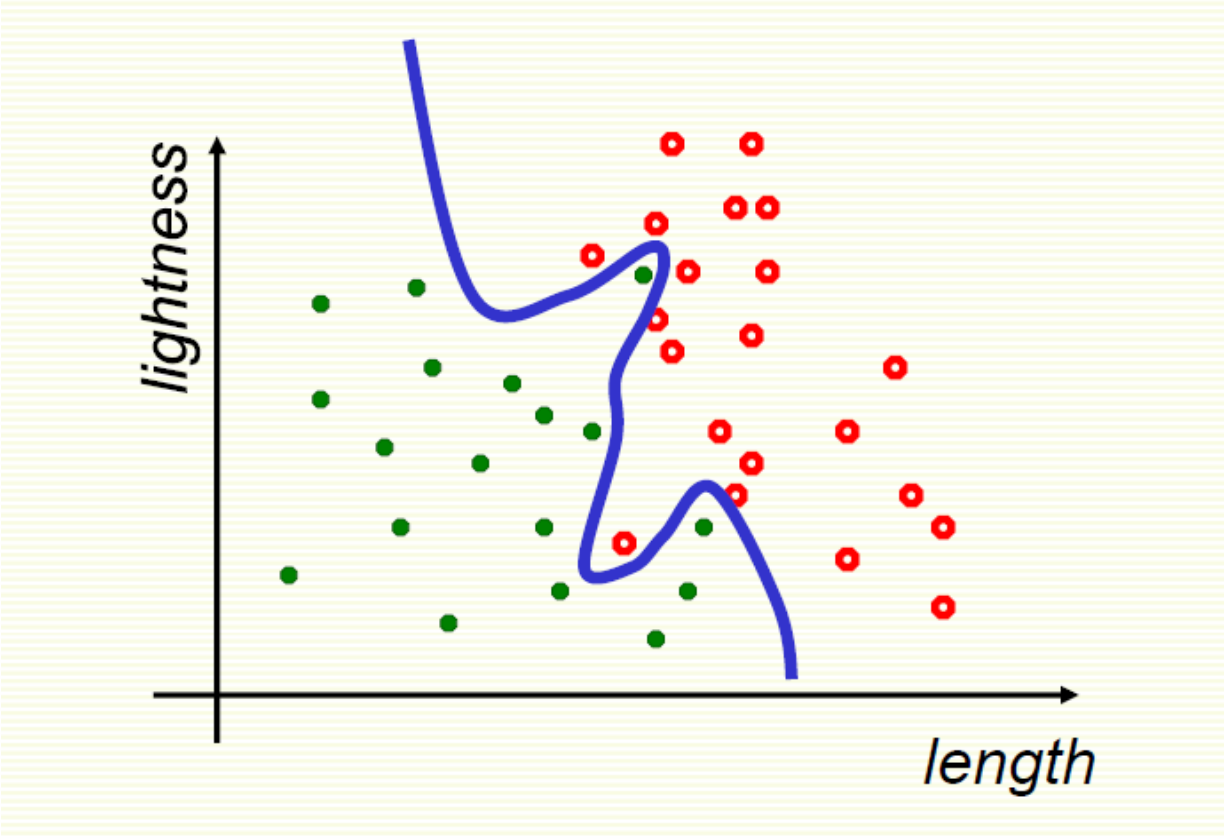
- Decision boundary (wiggly) with 0% classification error

Example : Fish sorting – On New Data



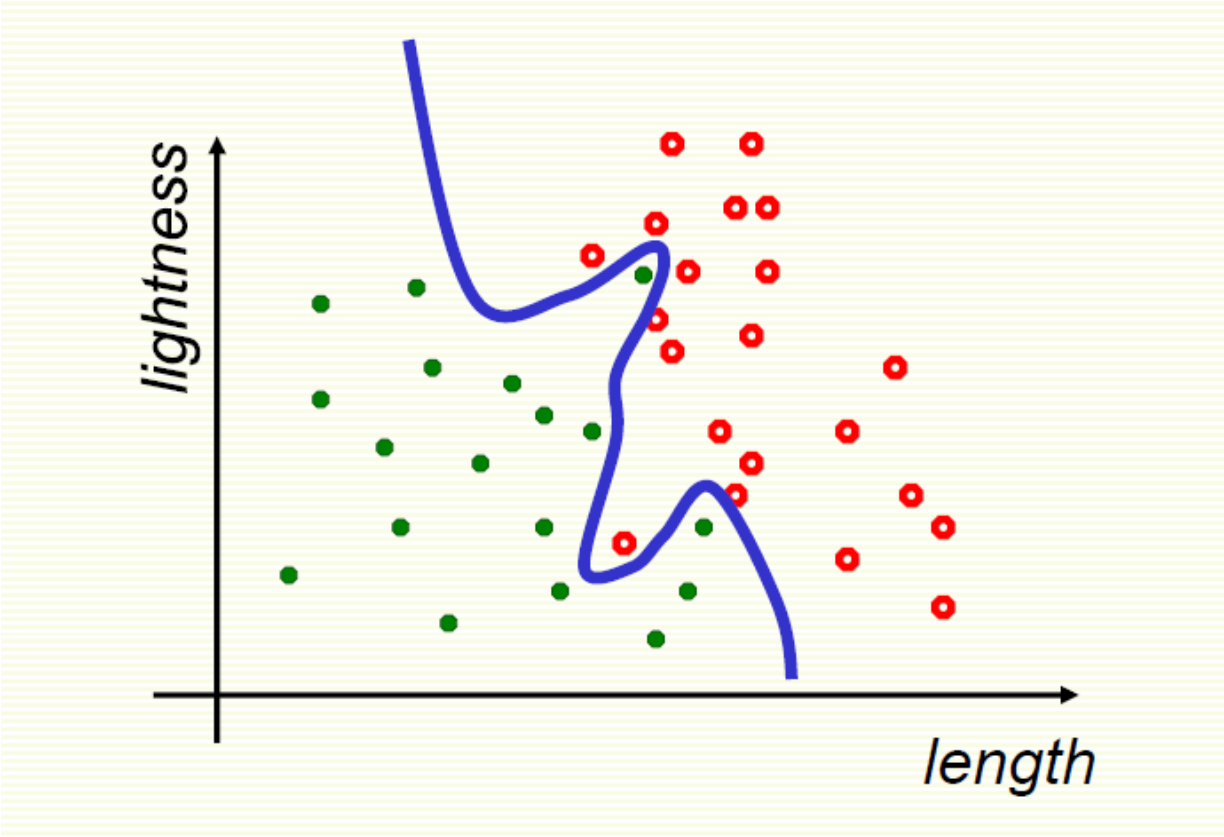
- The goal is for classifier to perform well on new data
- Test “wiggly” classifier on new data: 25% error

Example : Fish sorting – What went wrong?



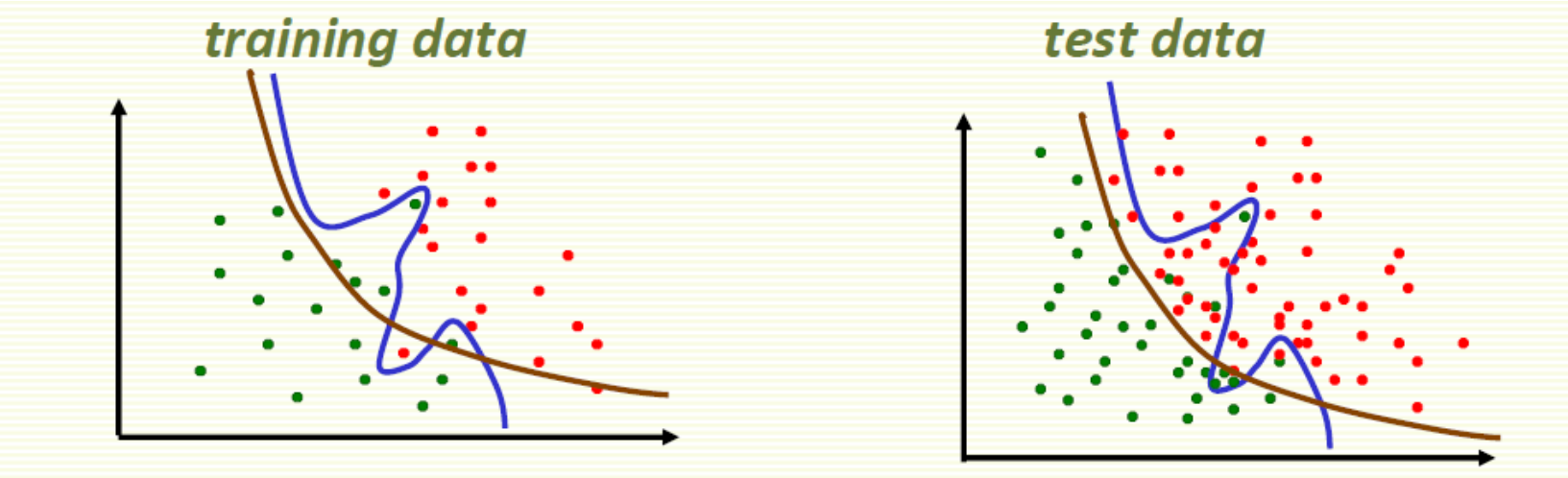
- Have only a limited amount of data for training
- Should ensure decision boundary does not adapt too closely to the particulars of training data, but grasps the “big picture”
- Complex boundaries **overfit** data, i.e. too tuned to training data

Example : Fish sorting – What went wrong?



- Have only a limited amount of data for training
- Should ensure decision boundary does not adapt too closely to the particulars of training data, but grasps the “big picture”
- Complex boundaries **overfit** data, i.e. too tuned to training data

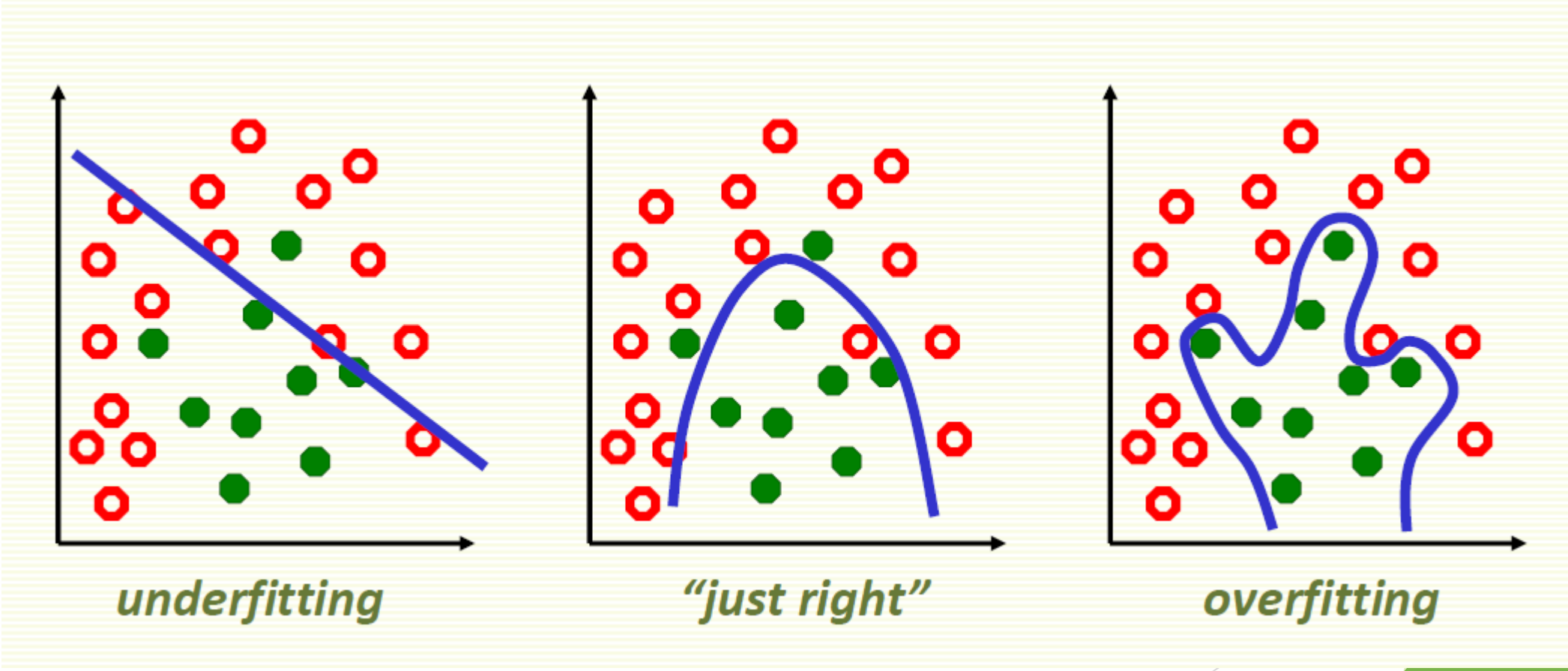
Generalization



- The ability to produce correct outputs on previously unseen examples is called **generalization**
- The big question of learning theory: how to get good generalization with a limited number of examples
- Intuitive idea: favour simpler classifiers
- William of Occam (1284-1347): “entities are not to be multiplied without necessity”
- Simpler decision boundary may not fit ideally to the training data but tends to generalize better to new data

Underfitting -> Overfitting

- You can also **underfit** the data



Sketch of Supervised Machine Learning

- Chose a function **f(x,w)**
  - **w** are tunable weights
  - **x** is the input sample
  - **f(x,w)** should output the correct class of sample **x**
  - use labeled samples to tune weights **w** so that **f(x,w)** give the correct label for sample **x**

- Which function **f(x,w)** do we choose?
  - different choices will lead to decision boundaries of different complexities
  - has to be expressive enough to model our problem well, i.e. to avoid ***underfitting***
  - yet not to complicated to avoid ***overfitting***
  - other issues, computational requirements, speed…

- **f(x,w)** sometimes called *learning machine*

Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Networks)
  - Support Vector Machines (SVM)

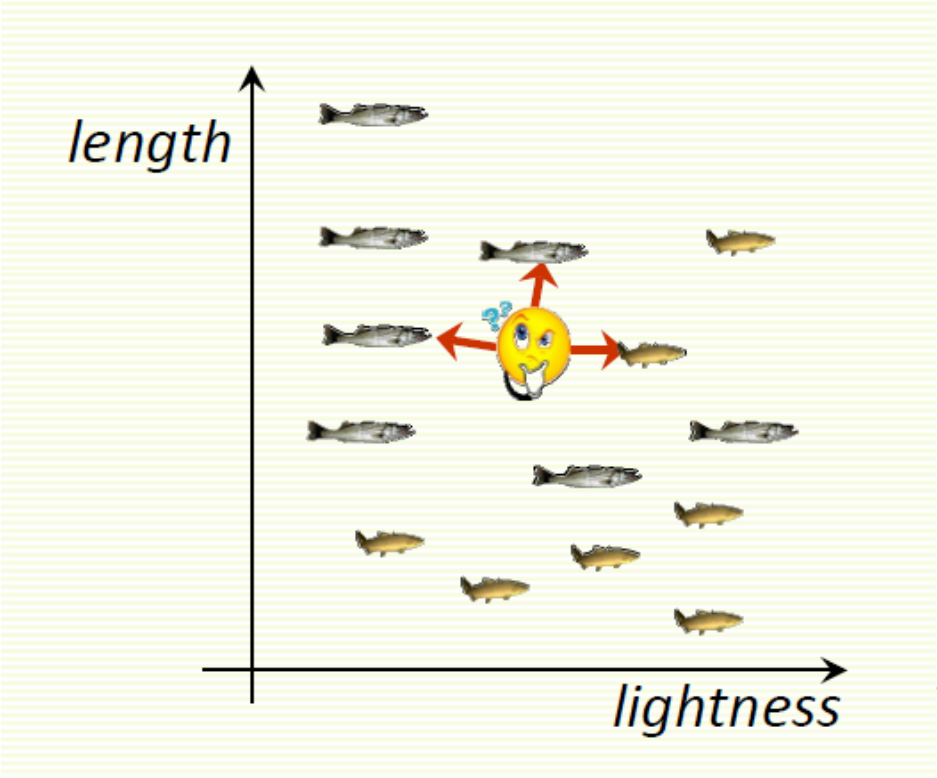
Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Networks)
  - Support Vector Machines (SVM)

k-Nearest Neighbors (kNN)

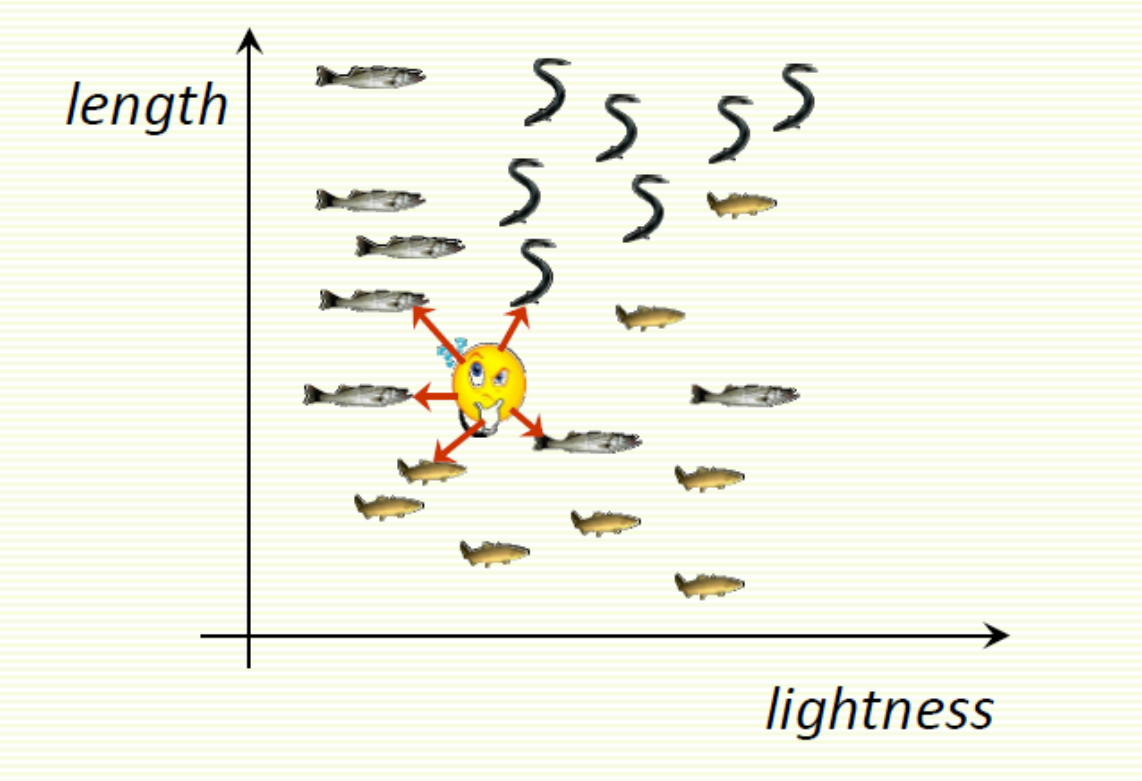
- kNN - the simplest classifier on earth
- Classify an unknown example with the most common class among k closest examples
  - “tell me who your neighbors are, and I’ll tell you who you are”

- Example:
  - k = 3
  - 2 sea bass, 1 salmon
  - Classify as sea bass



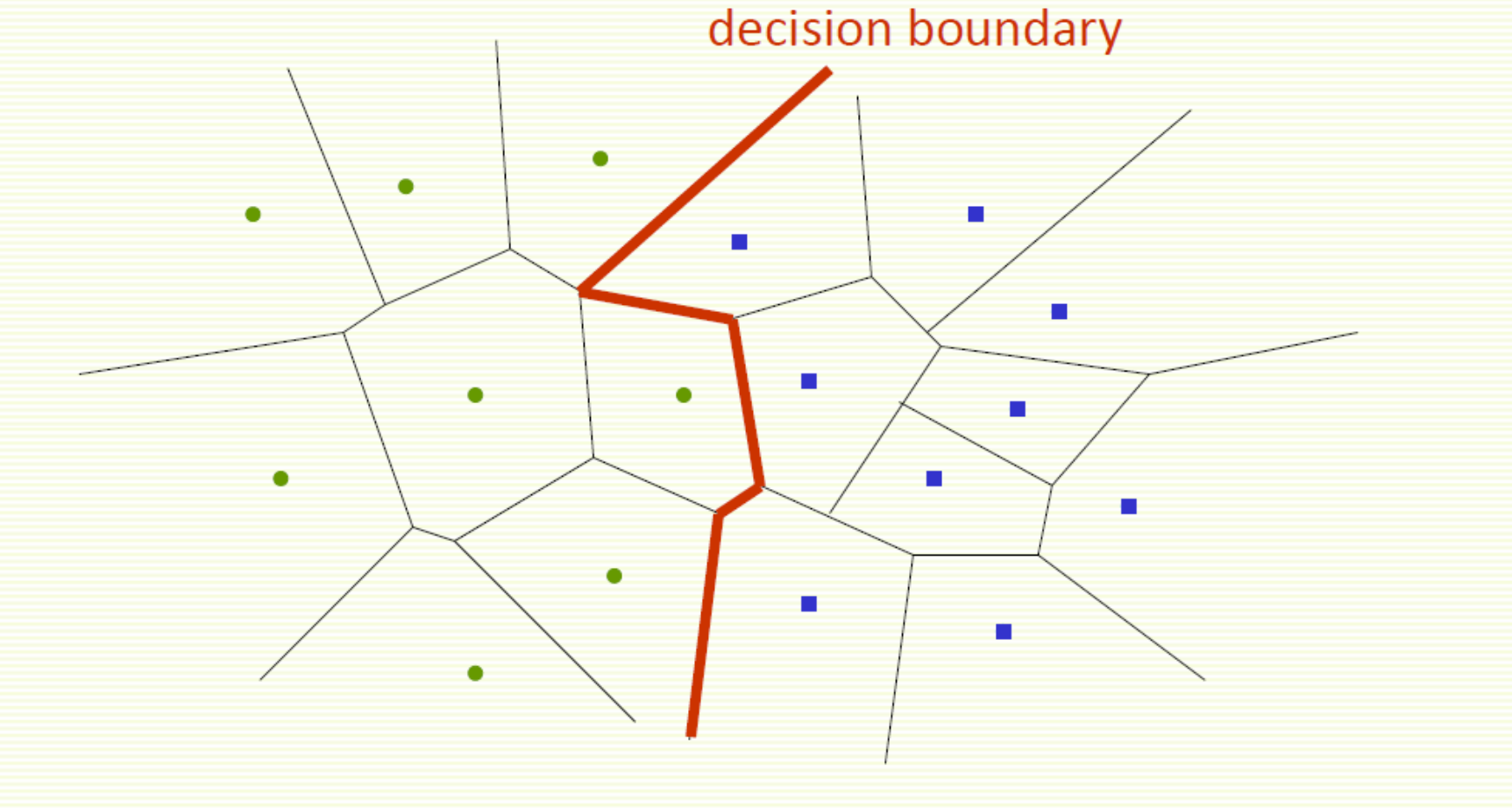
kNN - Multiple Classes

- Easy to implement for multiple classes
- Example for ***k*** = 5
  - 3 fish species: salmon, sea bass, eel
  - 3 sea bass, 1 eel, 1 salmon ⇒ classify as sea bass



kNN - Visualisation

- For k=1 Voronoi tesselation is useful



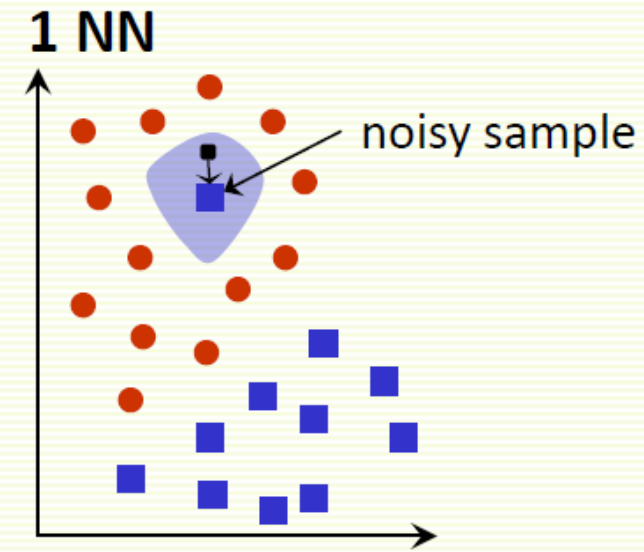
kNN: How to Choose k?

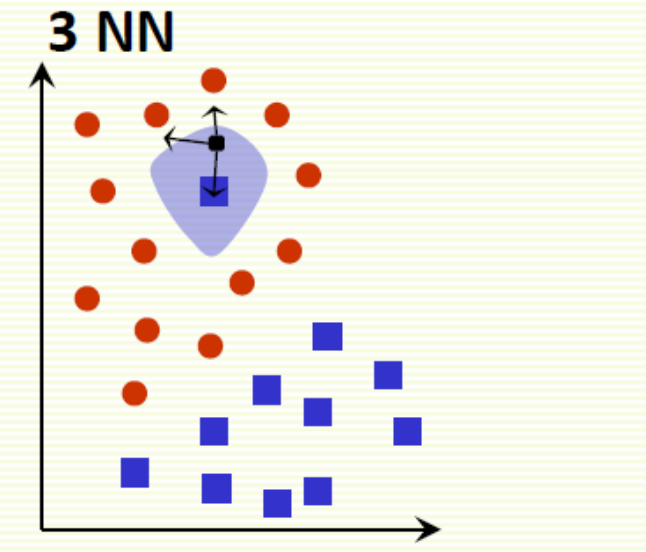
- In theory, if infinite number of samples available, the larger is ***k***, the better is classification

- But the caveat is that all ***k*** neighbors have to be close
  - Possible when infinite # samples available
  - Impossible in practice since # samples is finite

kNN: How to Choose k?

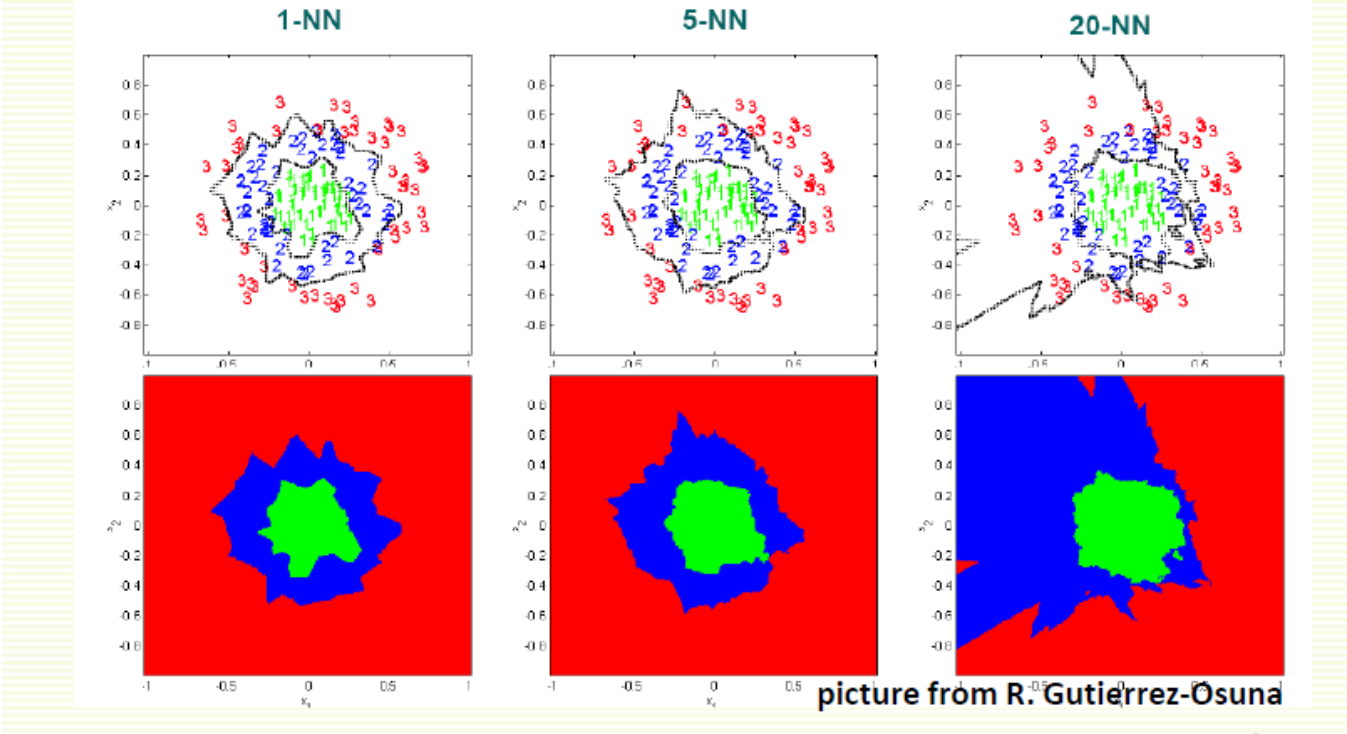
- Rule of thumb is ***k*** = sqrt(***n***), ***n*** is number of examples
  - interesting theoretical properties
- In practice, ***k*** = 1 is often used for efficiency, but can be sensitive to “noise”





kNN: How to Choose k?

- Larger ***k*** gives smoother boundaries, better for generalization
  - But only if *locality* is preserved. Locality is not preserved if end up looking at samples too far away, not from the same class.
- Interesting theoretical properties if ***k*** < sqrt(***n***), ***n*** is # of examples
- Can choose ***k*** through a method called cross-validation



picture from R. Gutierrez-Osuna

kNN: How Well does it Work?

- kNN is simple and intuitive, but does it work?
- Theoretically, the best error rate is the Bayes rate ***E****
  - Bayes error rate is the best (smallest) error rate a classifier can have, for a given problem, but we do not study it in this course
- Assume we have an unlimited number of samples
- kNN leads to an error rate greater than ***E****
- But even for ***k*** =1, as $n \rightarrow \infty$, it can be shown that kNN error rate is smaller than ***2E****
- As we increase ***k***, the upper bound on the error gets better, that is the error rate (as $n \rightarrow \infty$) for the ***kNN*** rule is smaller than ***cE****,with smaller ***c*** for larger ***k***
- **If we have lots of samples, kNN works well !**
  - Imagine that you have access to all past, current and future samples…

kNN - Summary

- Advantages
  - Can be applied to the data from any distribution
    - for example, data does not have to be separable with a linear boundary
  - Very simple and intuitive
  - Good classification if the number of samples is large enough

- Disadvantages
  - Choosing ***k*** may be tricky
  - Test stage is computationally expensive
    - No training stage, all the work is done during the test stage
    - This is actually the opposite of what we want. Usually we can afford training step to take a long time, but we want fast test step
  - Need large number of samples for accuracy

Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Networks)
  - Support Vector Machines (SVM)

Naïve Bayes Classifier

- Simple ("naïve") classification method based on Bayes rule

$$p(c_j \mid d) = \frac{p(d \mid c_j)\, p(c_j)}{p(d)}$$

$p(c_j \mid d)$ = probability of instance d being in class c_j,
This is what we are trying to compute

$p(d \mid c_j)$ = probability of generating instance d given class c_j,
We can imagine that being in class c_j, causes you to have feature d with some probability

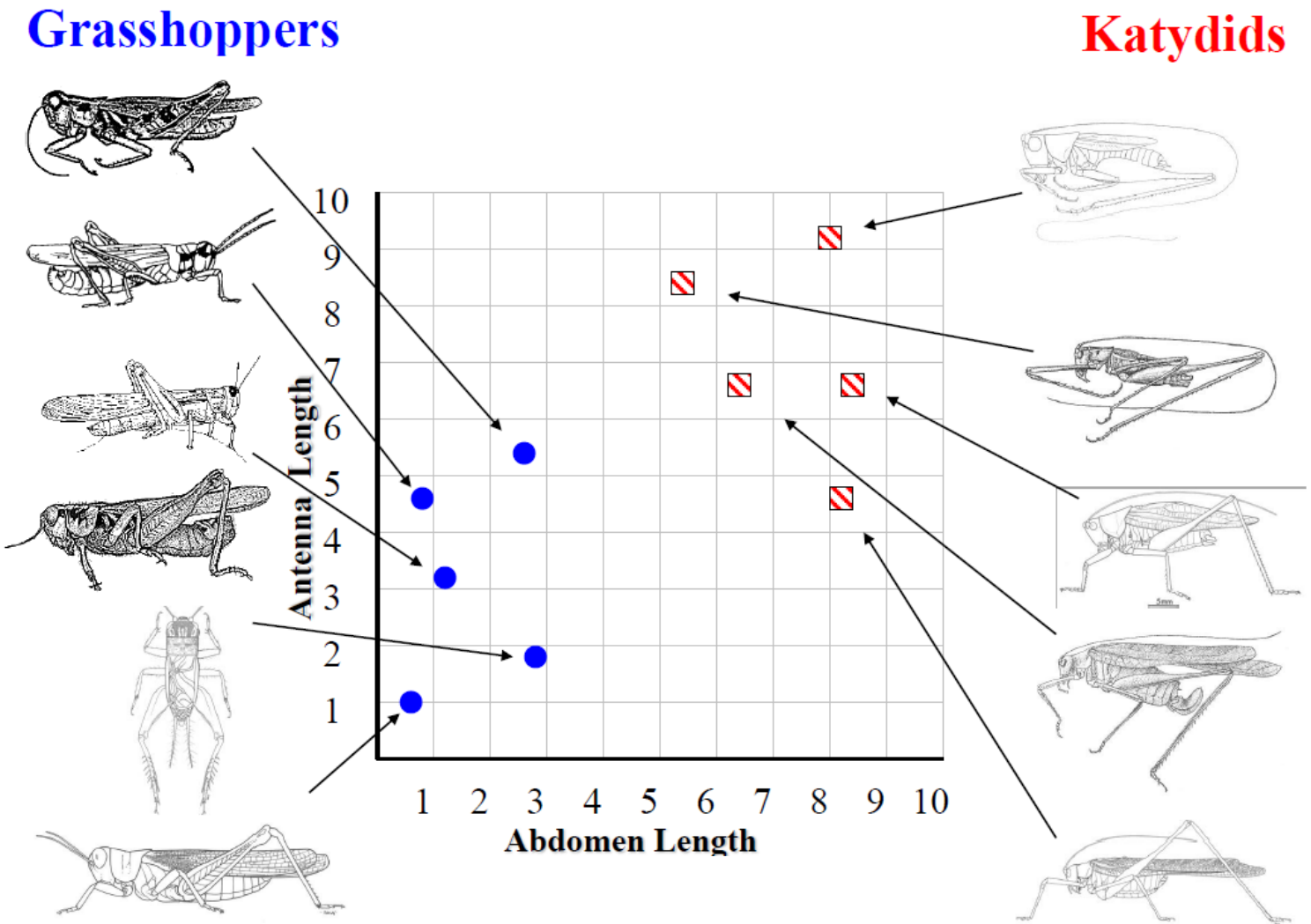
$p(c_j)$ = probability of occurrence of class c_j,
This is just how frequent the class c_j, is in our database

$p(d)$ = probability of instance d occurring

This can actually be ignored, since it is the same for all classes

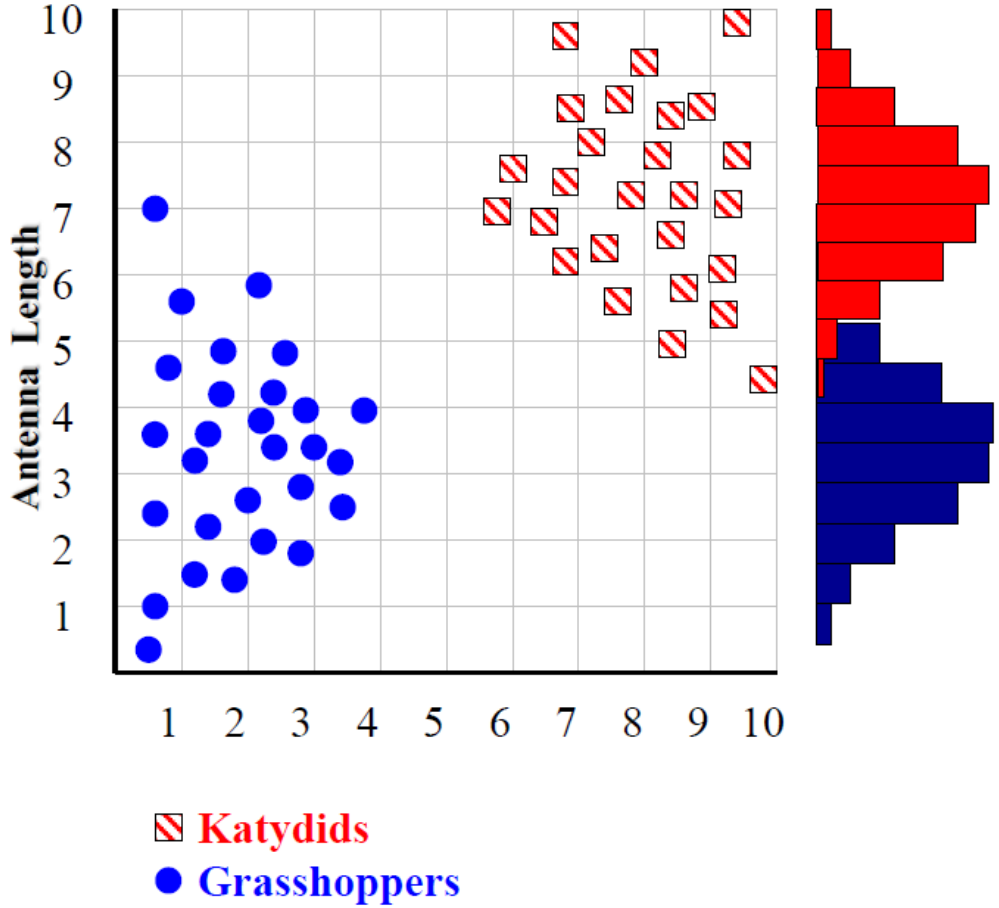
Thomas Bayes
1702 - 1761

Naïve Bayes Classifier - Example



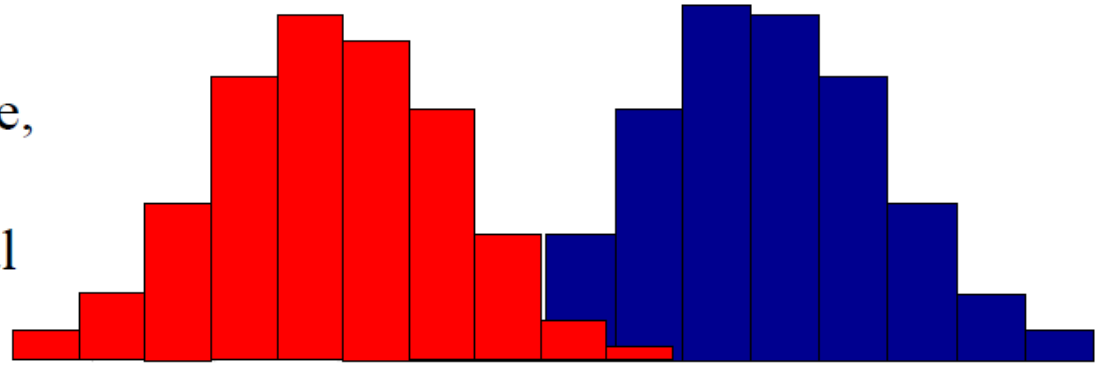
Naïve Bayes Classifier - Example

- With a lot of data, we can build a histogram. Let us just build one for “Antenna Length” for now…

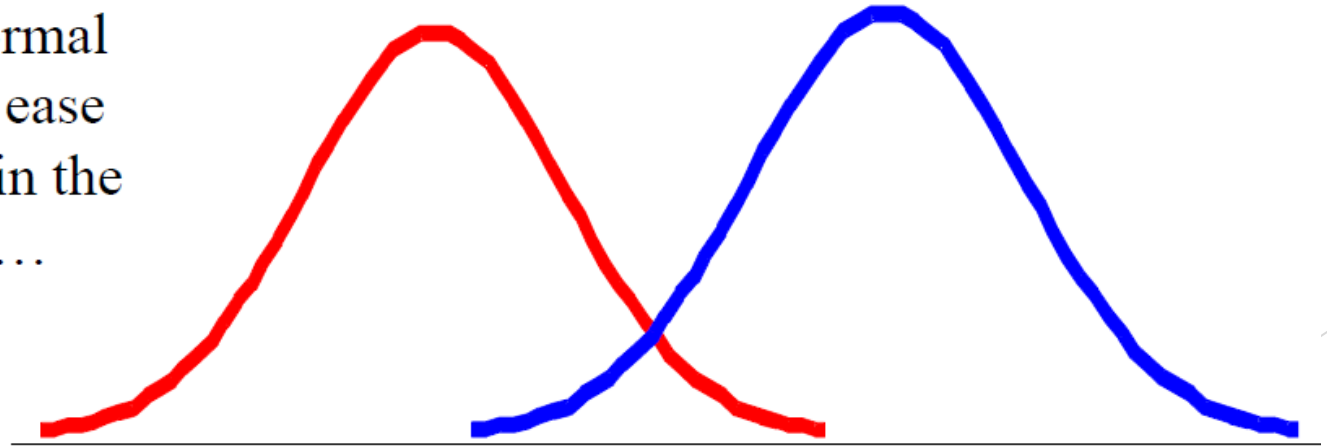


Naïve Bayes Classifier - Example

We can leave the histograms as they are, or we can summarize them with two normal distributions.

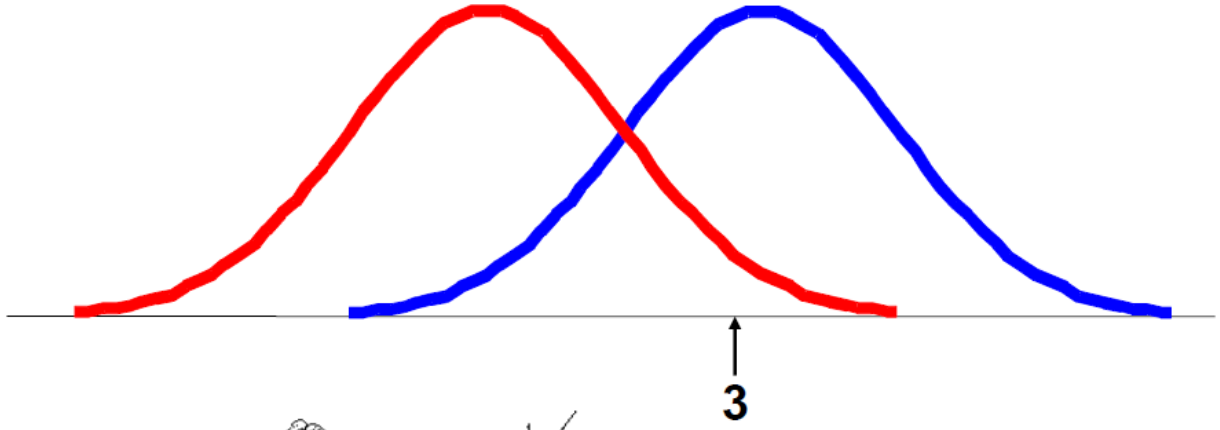


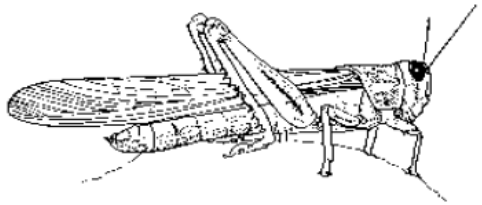
Let us us two normal distributions for ease of visualization in the following slides…



Naïve Bayes Classifier - Example

- We want to classify an insect we have found. Its antennae are 3 units long. How can we classify it?
- We can just ask ourselves, give the distributions of antennae lengths we have seen, is it more *probable* that our insect is a **Grasshopper** or a **Katydid** (Bush cricket).
- There is a formal way to discuss the most *probable* classification…
  - $p(c_j \mid d)$ = probability of class c_j, *given* that we have observed d.

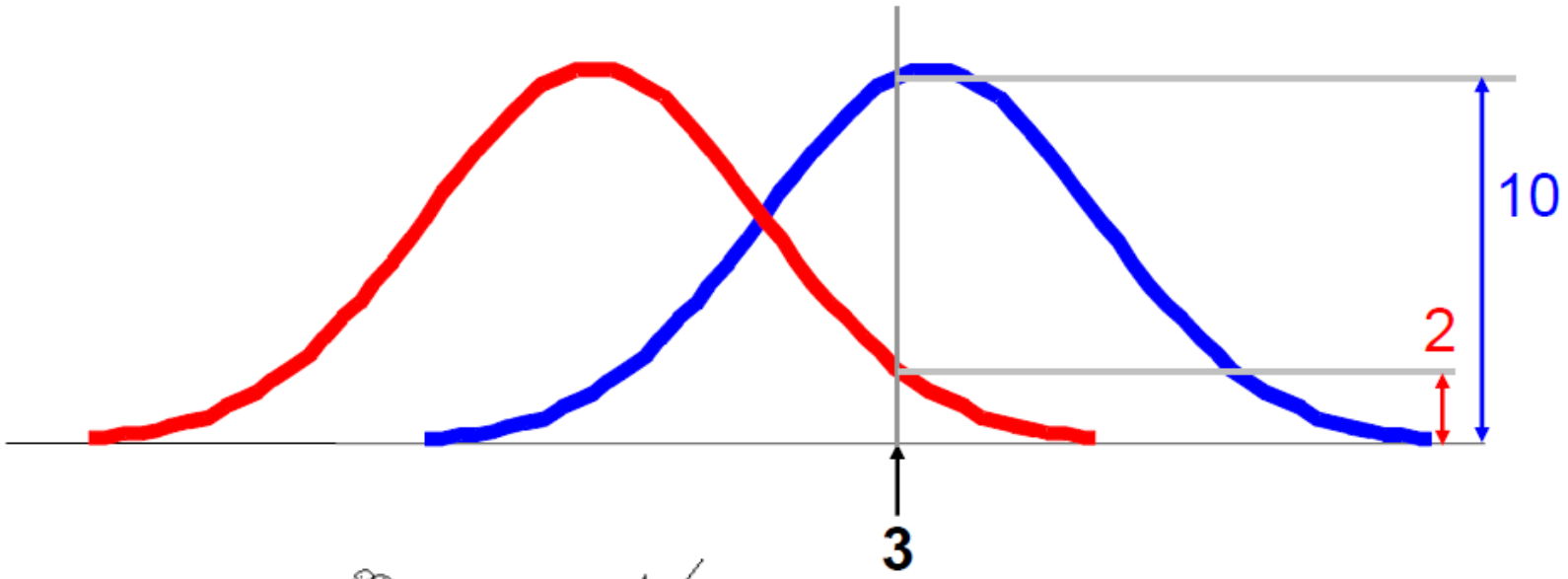


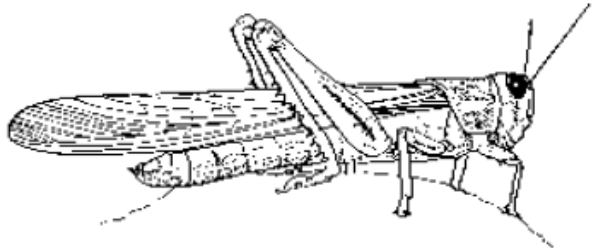


Antennae length is **3**

Naïve Bayes Classifier - Example

- P(**Grasshopper**| **3**) = 10/ (10+2)= 0.833
- P(**Katydid**| **3**) = 2/ (10+ 2)= 0.166

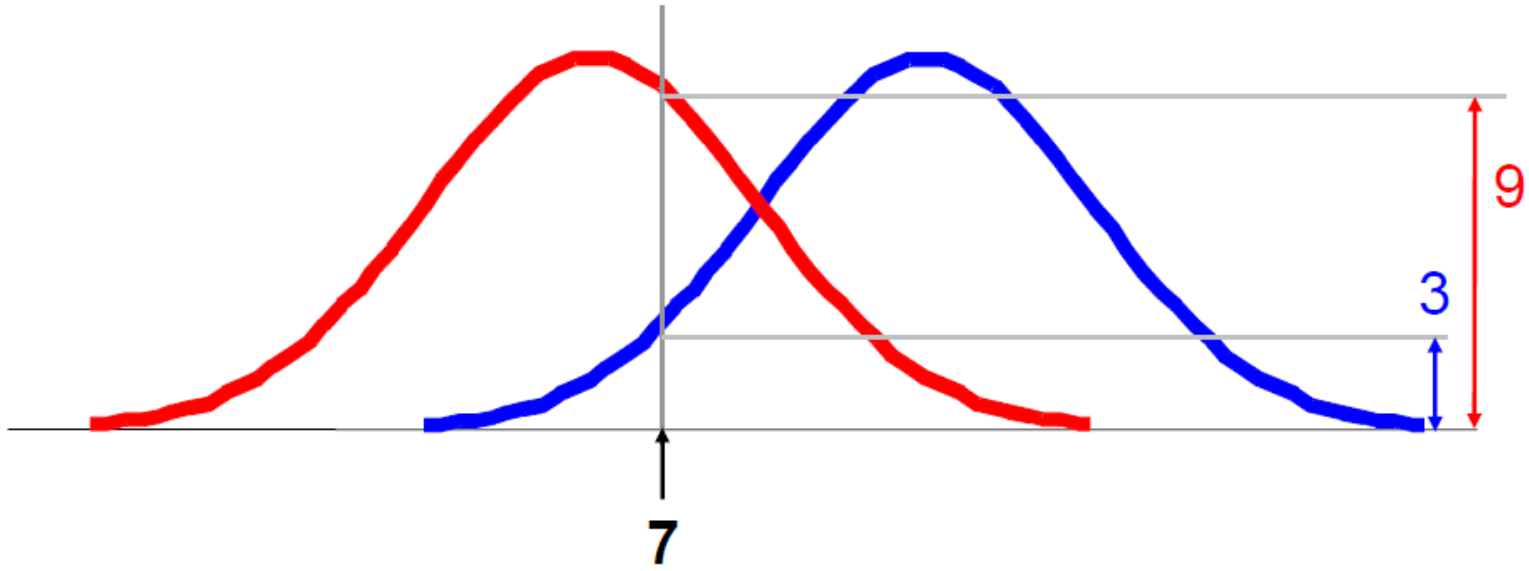




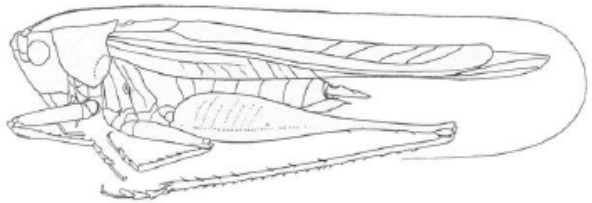
Antennae length is **3**

Naïve Bayes Classifier - Example

- P(**Grasshopper**| **7**) = 3/ (3+9)= 0.250
- P(**Katydid**| **7**) = 9/ (3+ 9)= 0.750

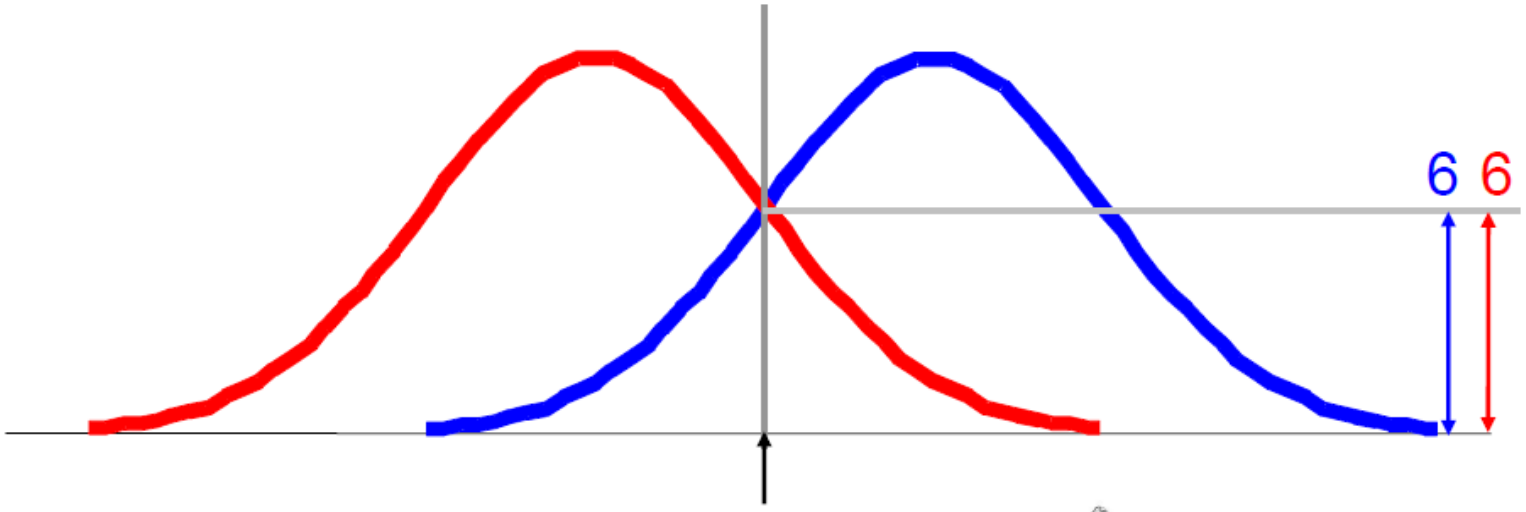


Antennae length is **7**

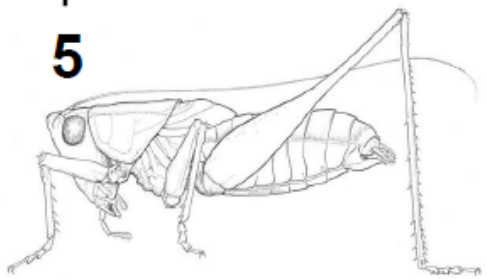


Naïve Bayes Classifier - Example

- P(**Grasshopper** | **5**) = 6/ (6+6)= 0.500
- P(**Katydid** | **5**) = 6/ (6+ 6)= 0.500



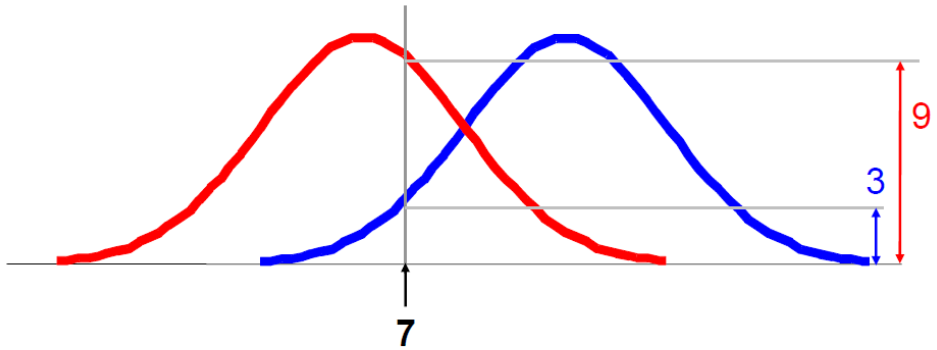
Antennae length is **5**



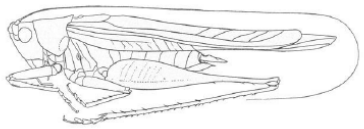
Naïve Bayes Classifier

- A basic example of the Naïve Bayes classifier, also called:
  - Idiot Bayes
  - Simple Bayes

- *Find out the probability of the previously unseen instance belonging to each class, then simply pick the most probable class.*



Antennae length is **7**



Naïve Bayes Classifier

- Bayesian classifiers use **Bayes theorem**, which says

$$p(c_j \mid d) = \frac{p(d \mid c_j)\, p(c_j)}{p(d)}$$

Example: c_j = "bush cricket",

d = "*antennae length = 3 cm*"

- $p(c_j \mid d)$ = probability of instance d being in class c_j,
  This is what we are trying to compute
- $p(d \mid c_j)$ = probability of generating instance d given class c_j,
  We can imagine that being in class c_j, causes you to have feature d with some probability
- $p(c_j)$ = probability of occurrence of class c_j,
  This is just how frequent the class c_j, is in our database
- $p(d)$ = probability of instance d occurring

  This can actually be ignored, since it is the same for all classes

Naïve Bayes Classifier – Multiple Features

- So far we have only considered Bayes Classification when we have one attribute (the "*antennae length*").
- But we may have many features. ("*antennae length*", "*colour*", "weight")
- How do we use all the features?

Naïve Bayes Classifier – Multiple Features

- Assume attributes have independent distributions, and thereby estimate

$$p(c_j \mid d) = \frac{p(d \mid c_j)\, p(c_j)}{p(d)}$$

c_j = "bush cricket"

d = {"*antennae length = 3 cm*",
"*colour= green*", "weight= 25 grams")

$$p(d|c_j) = p(d_1|c_j) * p(d_2|c_j) * \ldots * p(d_n|c_j)$$

The probability of class c_j generating instance d, equals….

The probability of class c_j generating the observed value for feature 1, multiplied by..

The probability of class c_j generating the observed value for feature 2, multiplied by..

Naïve Bayes Classifier - Summary

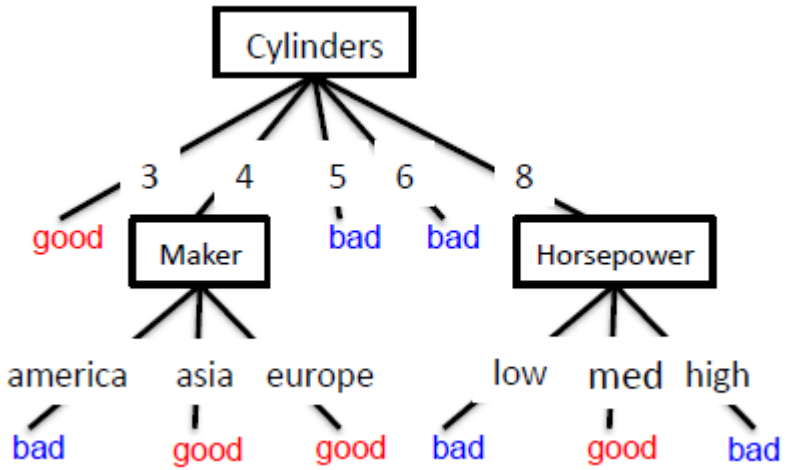
- Advantages
  - Fast to train (single scan)
  - Fast to classify
  - Not sensitive to irrelevant features
  - Handles real and discrete data
- Disadvantages
  - Assumes independence of features

Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Networks)
  - Support Vector Machines (SVM)

Decision Tree

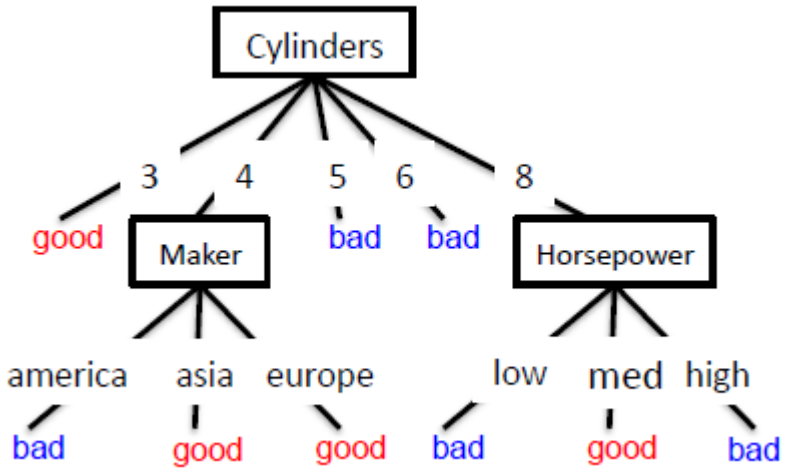
- a hierarchical tree structure used as a classifier
- based on a series of questions (or rules) about the attributes of the class.
- the attributes of the classes can be any type of variables from binary, nominal, ordinal, and quantitative values,



Example: Predict fuel efficiency of a car

Decision Tree

- A **decision tree** has 2 kinds of **nodes**
  1. Each **leaf node** has a class label, determined by majority vote of training examples reaching that leaf.
  2. Each **internal node** is a question on features. It branches out according to the answers.



Example: Predict fuel efficiency of a car

Decision Tree - Choosing the attributes

- How do we find a decision tree that agrees with the training data?
- Could just choose a tree that has one path to a leaf for each example - but this just memorizes the observations (assuming data are consistent)
- We want it to generalize to new examples
- deally, best attribute would partition the data into positive and negative examples

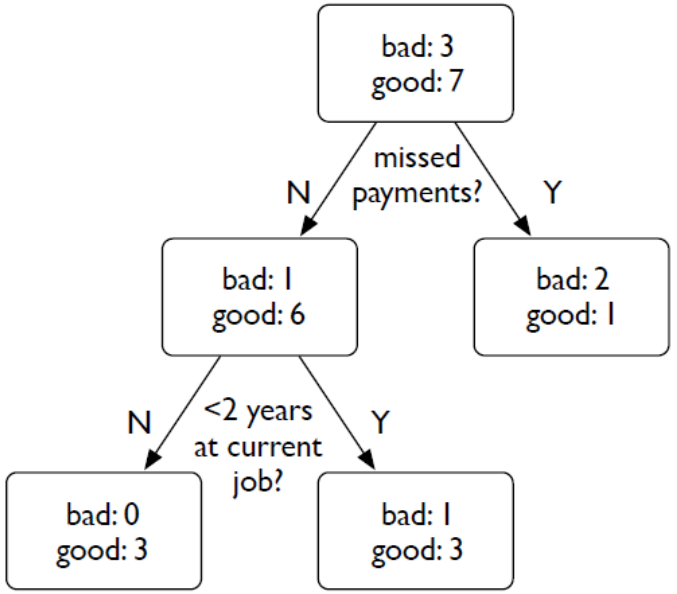
- Learning decision trees is hard. Finding optimal tree for arbitrary data is NP-hard.
- Strategy (greedy):
  - choose attributes that give the best partition first
  - want correct classification with fewest number of tests

Decision Tree - Choosing the attributes

- How do we which attribute or value to split on?
- When should we stop splitting?
- What do we do when we can't achieve perfect classification?
- What if tree is too large? Can we approximate with a smaller tree?

Predicting credit risk

<2 years at current job?	missed payments?	defaulted?
N	N	N
Y	N	Y
N	N	N
N	N	N
N	Y	Y
Y	N	N
N	Y	N
N	Y	Y
Y	N	N
Y	N	N



Decision Tree – Basic Algorithm

- Basic algorithm for learning decision trees
  1. starting with whole training data
  2. select attribute or value along dimension that gives “best” split
  3. create child nodes based on split
  4. recurse on each child using child data until a stopping criterion is reached
     - all examples have same class
     - amount of data is too small
     - tree too large
- Central problem: How do we choose the “best” attribute?

Decision Tree - Measuring information

- A convenient measure to use is based on information theory.
- How much “information” does an attribute give us about the class?
  - attributes that perfectly partition should given maximal information
  - unrelated attributes should give no information

- Entropy H(Y) of a random variable Y

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

- More uncertainty, more entropy!

Decision Tree

- **High Entropy**
  - Y is from a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable

- **Low Entropy**
  - Y is from a varied (peaks and valleys) distribution
  - Histogram has many lows and highs
  - Values sampled from it are more predictable

Decision Tree

- Idea: select the attribute that decreases the entropy (uncertainty) the most after splitting

- Maximise the **Information Gain**

$$IG(X) = H(Y) - H(Y \mid X)$$

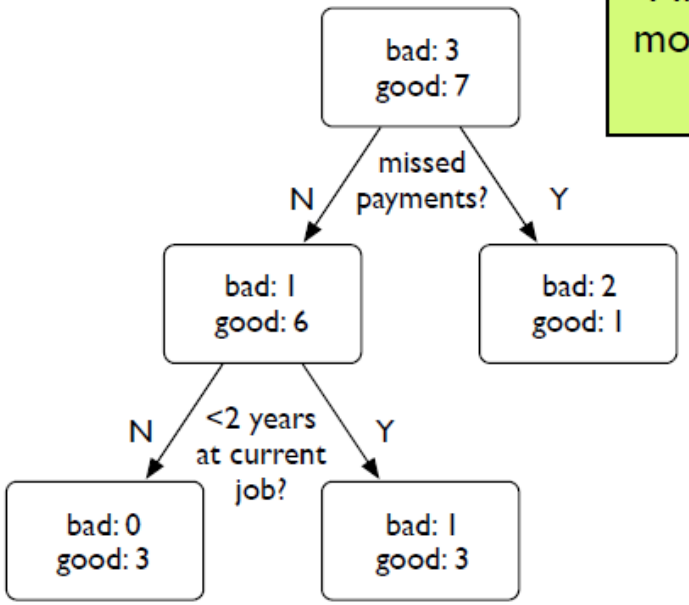
Decision Tree - Example

Predicting credit risk

<2 years at current job?	missed payments?	defaulted?
N	N	N
Y	N	Y
N	N	N
N	N	N
N	Y	Y
Y	N	N
N	Y	N
N	Y	Y
Y	N	N
Y	N	N

$$H(\text{defaulted}) - H(\text{defaulted}|< 2 \text{ years})$$
$$0.8813 - 0.8763 = 0.0050$$
$$H(\text{defaulted}) - H(\text{defaulted}|\text{missed})$$
$$0.8813 - 0.6897 = 0.1916$$

Missed payments are the most informative attribute about defaulting.



Decision Tree – Basic Algorithm

- Basic algorithm for learning decision trees
  1. starting with whole training data
  2. select attribute or value along dimension that maximises the **Information Gain**
  3. create child nodes based on split
  4. recurse on each child using child data until a stopping criterion is reached
     - all examples have same class
     - amount of data is too small
     - tree too large

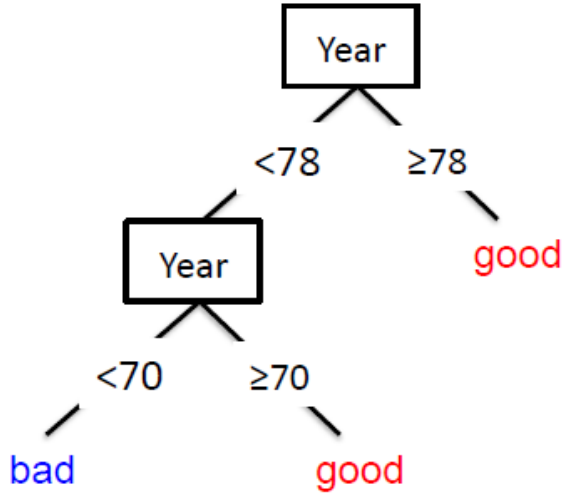
Decision Tree – Real-valued Inputs

- What should we do if some of the inputs are real-valued?

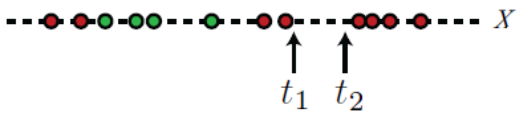
mpg	cylinders	displacemen	horsepower	weight	acceleration	modelyear	maker
good	4	97	75	2265	18.2	77	asia
bad	6	199	90	2648	15	70	america
bad	4	121	110	2600	12.8	77	europe
bad	8	350	175	4100	13	73	america
bad	6	198	95	3102	16.5	74	america
bad	4	108	94	2379	16.5	73	asia
bad	4	113	95	2228	14	71	asia
bad	8	302	139	3570	12.8	78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
good	4	120	79	2625	18.6	82	america
bad	8	455	225	4425	10	70	america
good	4	107	86	2464	15.5	76	europe
bad	5	131	103	2830	15.9	78	europe

Decision Tree – Real-valued Inputs

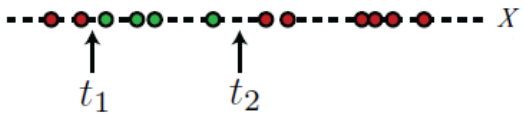
- Binary tree: split on attribute X at value t
  - One branch : X < t
  - Other branch : X ≥ t
- Allow repeated splits on same variable



- How to find t?
  - Search through all possible values of *t* (seems hard!)
  - But only a finite number of *t*'s are important:



  - Sort data according to X Into $\{x_1,\ldots,x_m\}$
  - Consider Split Points Of The Form $x_i + (x_{i+1} - x_i)/2$
  - Moreover, only splits between examples of different classes matter!



Decision Tree - Summary

- Decision trees are one of the most popular ML tools
- Easy to understand, implement and use
- Usually interpretable
- Computationally cheap (to solve heuristically)
- Presented for classification, can be used for regression and density estimation too
- Decision trees will overfit!
  - Must use tricks to find “simple trees”, e.g.,
    - Fixed depth/Early stopping
    - Pruning
    - Hypothesis testing

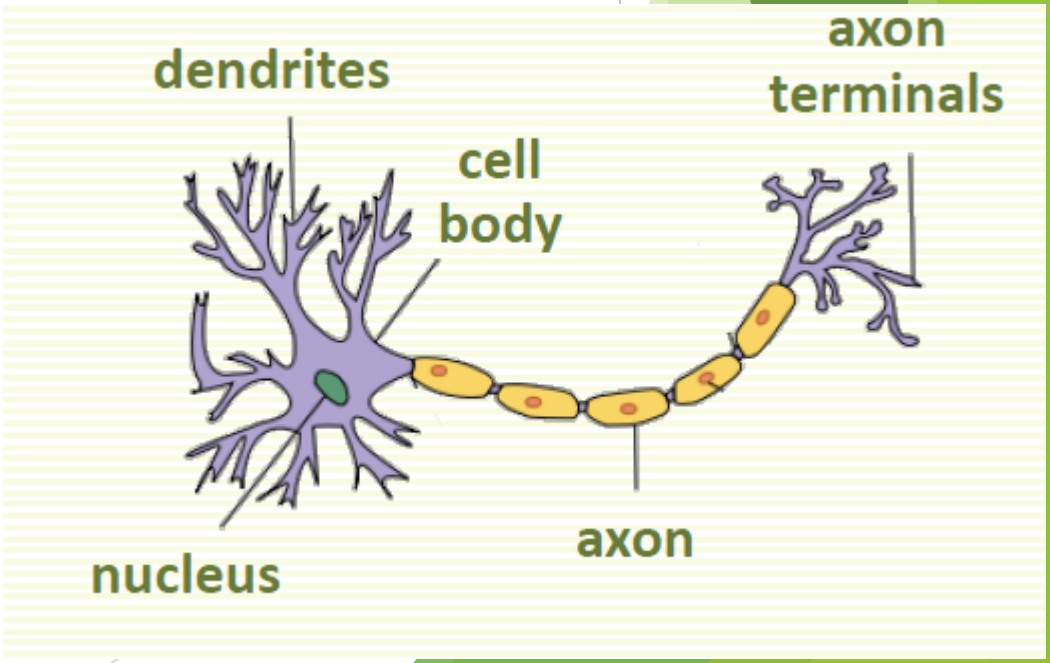
Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Networks)
  - Support Vector Machines (SVM)

Human Brain

- Neurons (or nerve cells) are special cells that process and transmit information by electrical signaling in brain and also spinal cord
- Human brain has around 10^{11} neurons
- A neuron connects to other neurons to form a network
- Each neuron cell communicates to anywhere from 1000 to 10,000 other neurons

- **cell body (**computational unit)
- **dendrites** (input wires)
  - receive inputs from other neurons
  - a neuron may have thousands of dendrites, usually short
- **axon** (output wire), sends signal to other neurons
- single long structure (up to 1 meter)
- splits in possibly thousands branches at the end, “axon terminals”



Artificial Neural Networks (ANN)

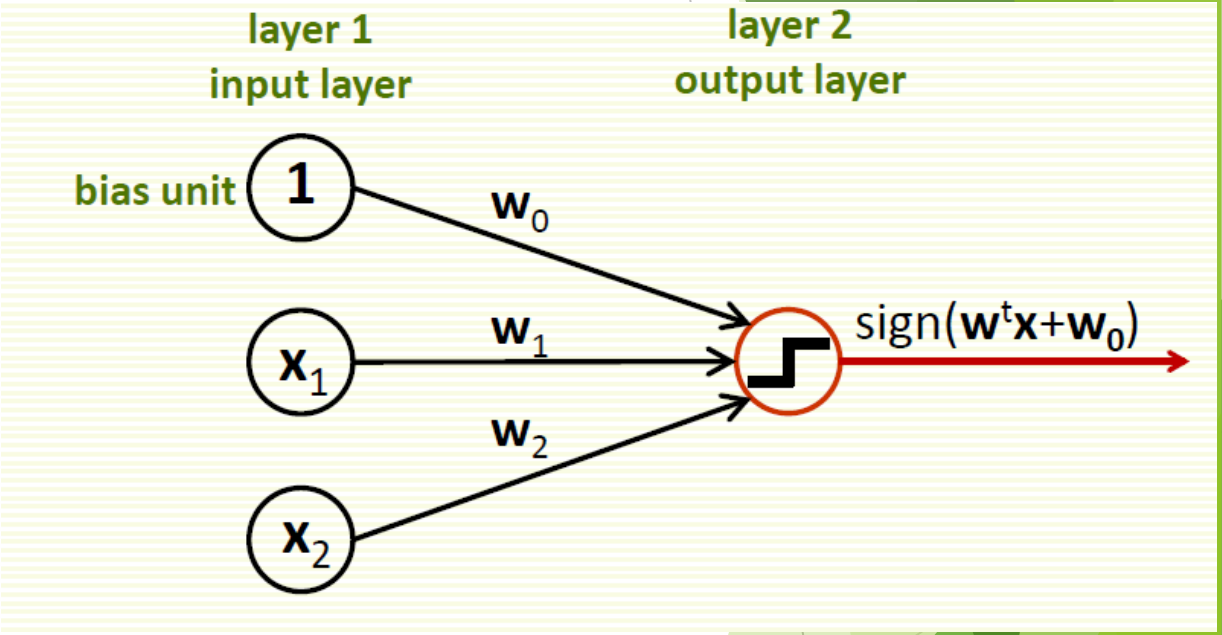
- A single neuron
  - Linear perceptron
  - Non-linear perceptron
  - Learning of a single perceptron
  - The power of a single perceptron
- Neural network: a network of neurons
  - Layers, hidden units
  - Learning of neural network: backpropagation
  - The power of neural network

Perceptron

- Perceptron - a mathematical model of a single neuron
- Oldest neural network still in use today (Rosenblatt 1958)
  - Simple algorithm to train the perceptron network
  - Proved convergence in linearly separable case

Perceptron

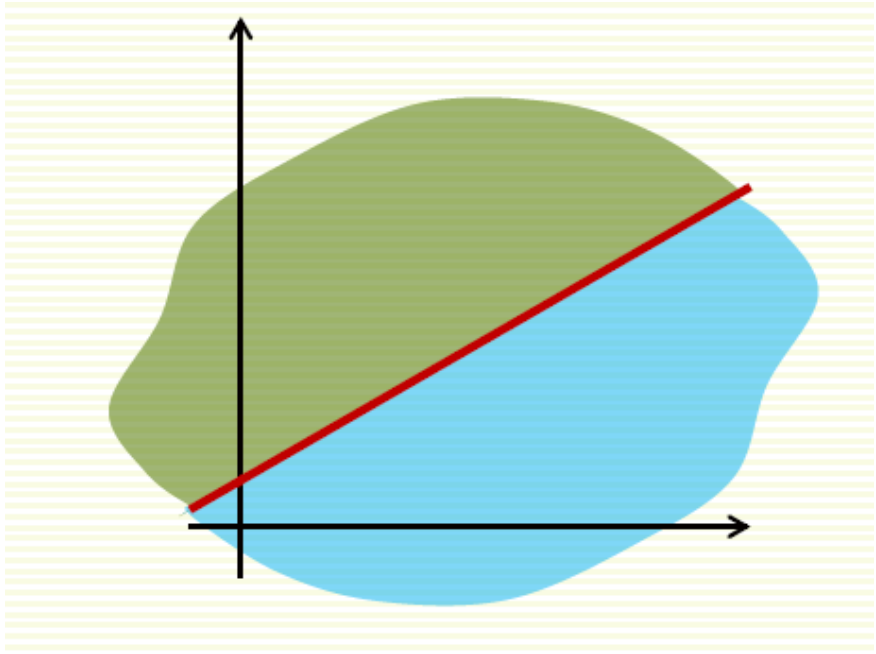
- Inputs are **feature values** (and a bias $x_0=1$)
- Each feature has a **weight**
- Sum is the **activation**
  - $\mathbf{f}(\mathbf{x}) = \text{sign}(\sum_i \mathbf{w}_i \mathbf{x}_i + \mathbf{w0})$
- If the activation is:
  - Positive, output +1
  - Negative, output -1



- Linear classifier **f**(**x**) is a single neuron "net"
- **h**(t) is also called an *activation function*

Perceptron - Decision Boundary

- Recall from linear algebra
  - Plane P defined by normal **n** and offset b.
  - (Signed) distance to P given by $\Sigma_i n_i x_i + b$
  - Set n=w and b=w_0
- Perceptron gives a linear **Decision Boundary** (hyperplane) where w defines the normal and w_0 the offset.



Perceptron - Training

The Perceptron Algorithm

- Start with zero weights (or random weights)
- For each training instance:
  - Classify ith sample with current weights

    $y_i = \text{sign}(w^T x_i)$

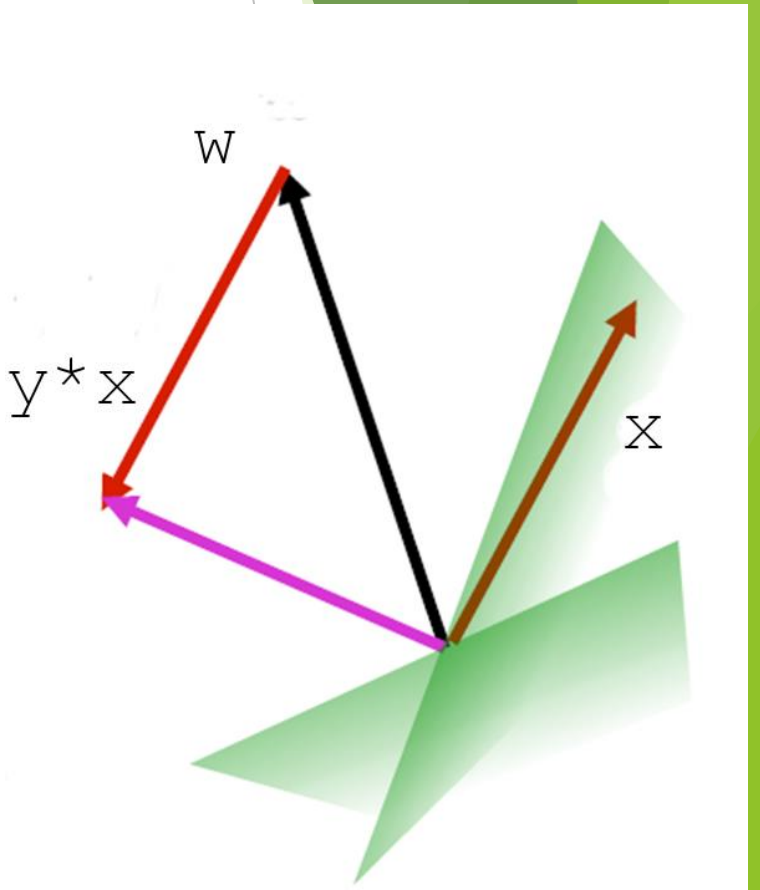
  - Compare with true label y_i^*

    If correct (i.e., y=y*), no change!

    If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

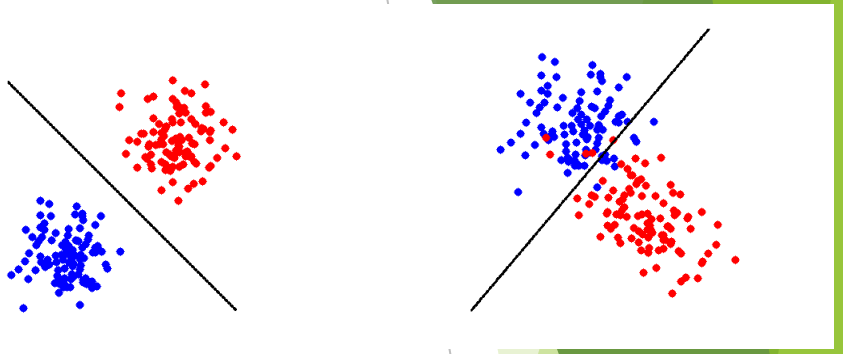
    $w_{new} = w + y_i^* x$

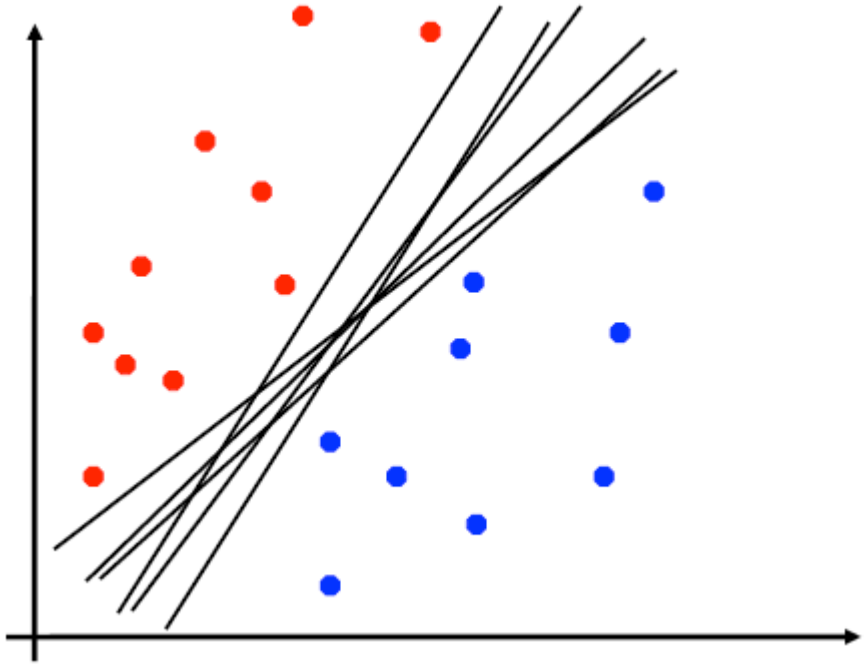
- Repeat until all x_i correctly classified



Perceptron - Training

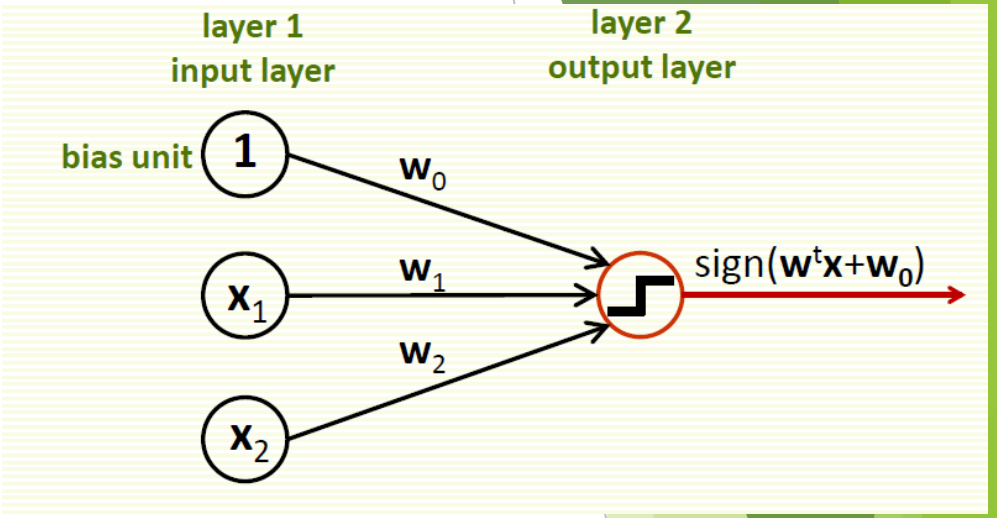
- Convergent if linearly separable
- Returns a separating hyperplane (which one?)





Perceptron - Summary

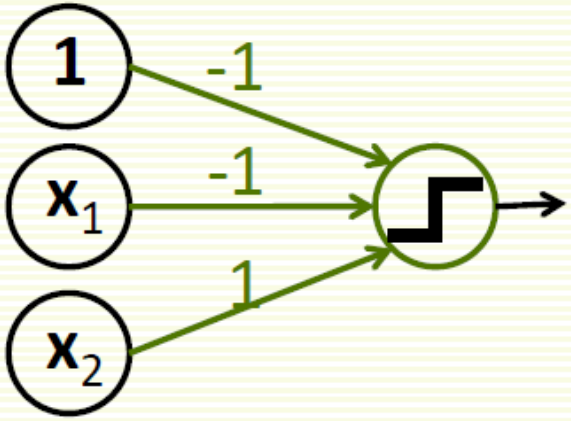
- Advantages
  - Very simple algorithm
  - Guaranteed convergence when linearly separable
  - Fast on test data
- Disadvantages
  - Fails when not linearly separable

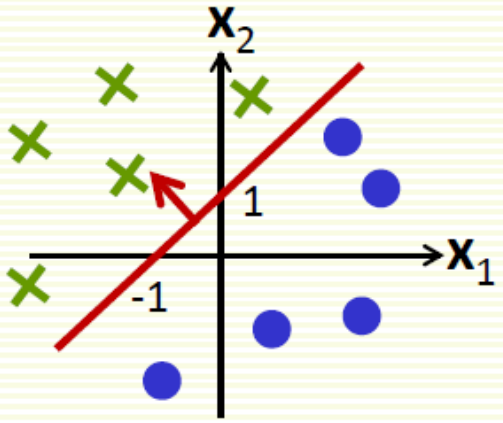


The perceptron algorithm either converges to global optimum or never converges.

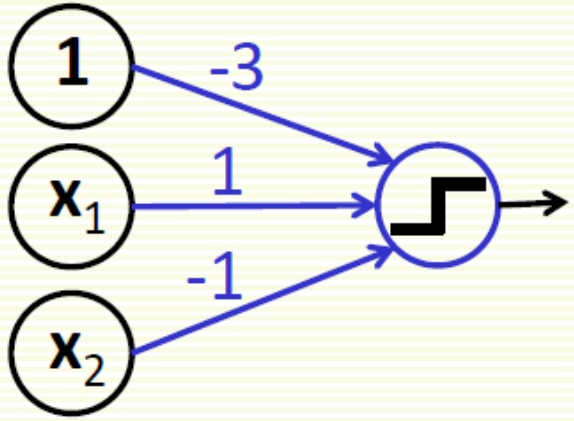
Multilayer Perceptron

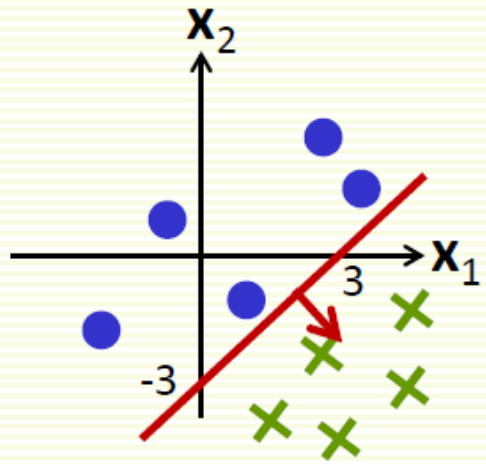
$-x_1 + x_2 - 1 > 0 \Rightarrow$ class 1





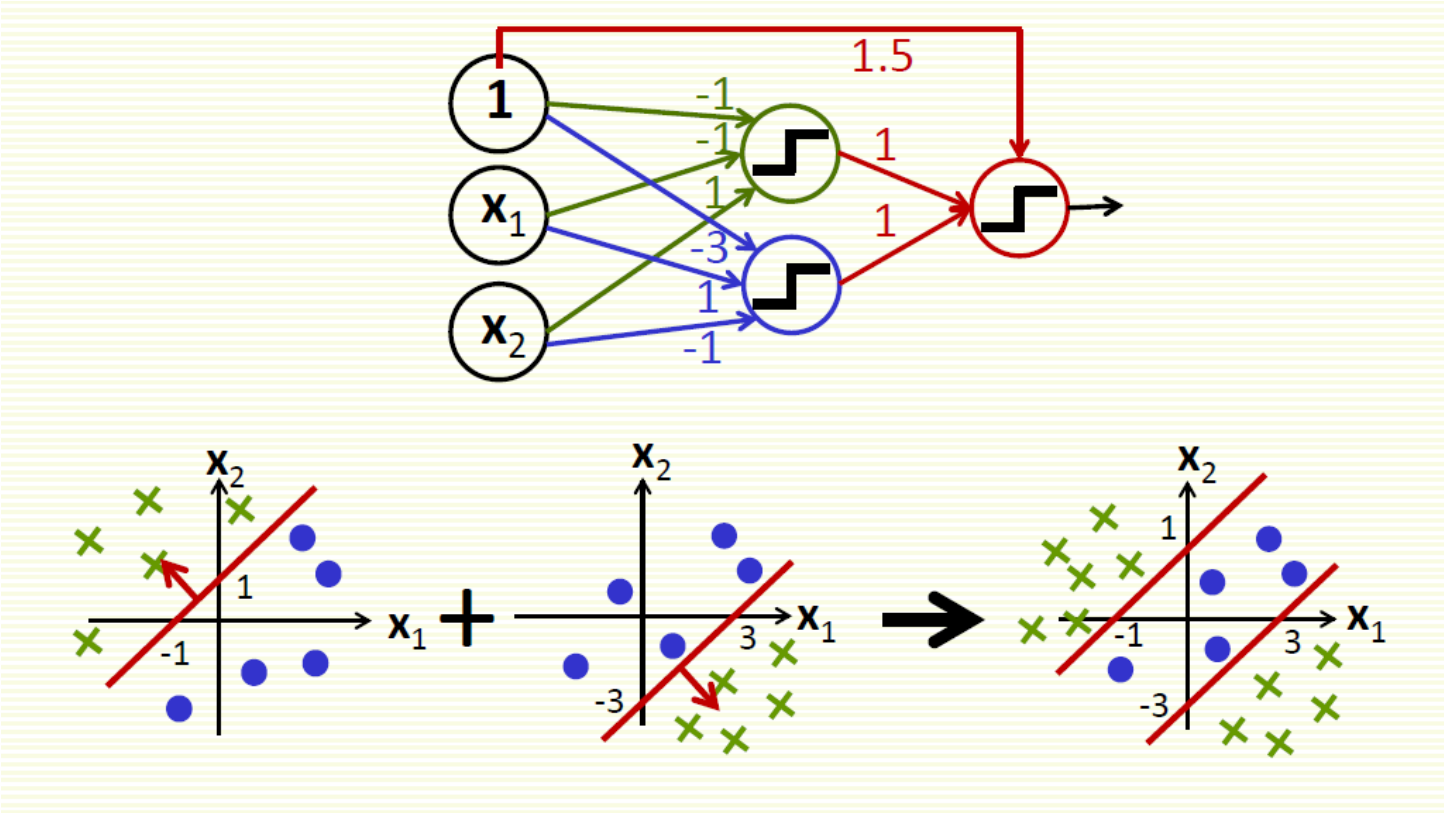
$x_1 - x_2 - 3 > 0 \Rightarrow$ class 1





Multilayer Perceptron

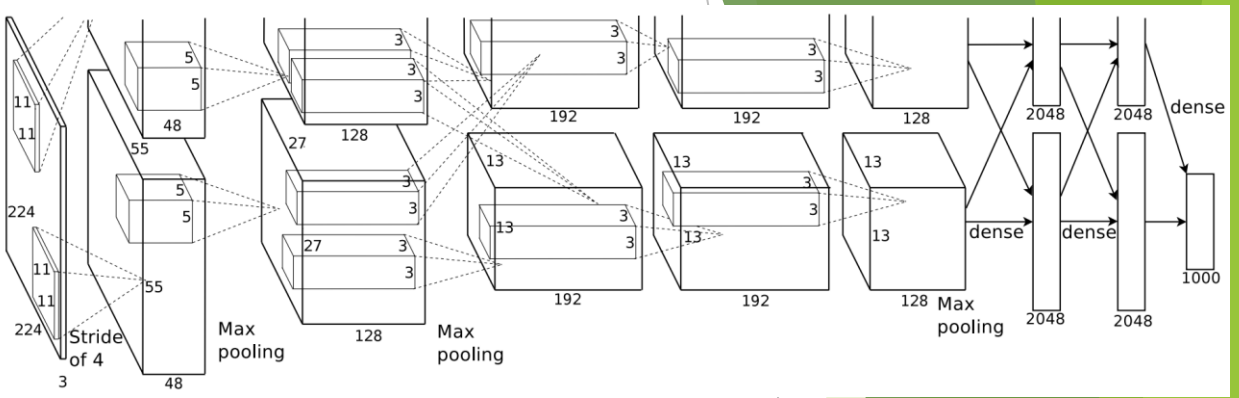
- Combine two Perceptrons into a 3 layer Neural Network
- Achieve arbitrarily complex decision regions
- Even not contiguous



Deep Learning

- Driving the latest AI revolution
- A result of increase in
  - computational power
  - amounts of available data

- Apple Siri
- IBM Watson
- Google self driving car
- Facebook user modelling
- and many more…



torch

Facebook AI Research

Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Network)
  - Support Vector Machines (SVM)

Support Vector Machines (SVM)

- Said to start in 1979 with Vladimir Vapnik's paper
- Major developments throughout 1990's
- Elegant theory
  - Has good generalization properties
- Have been applied to diverse problems very successfully in the last 10-15 years
- One of the most important developments in pattern recognition in the last 15 years

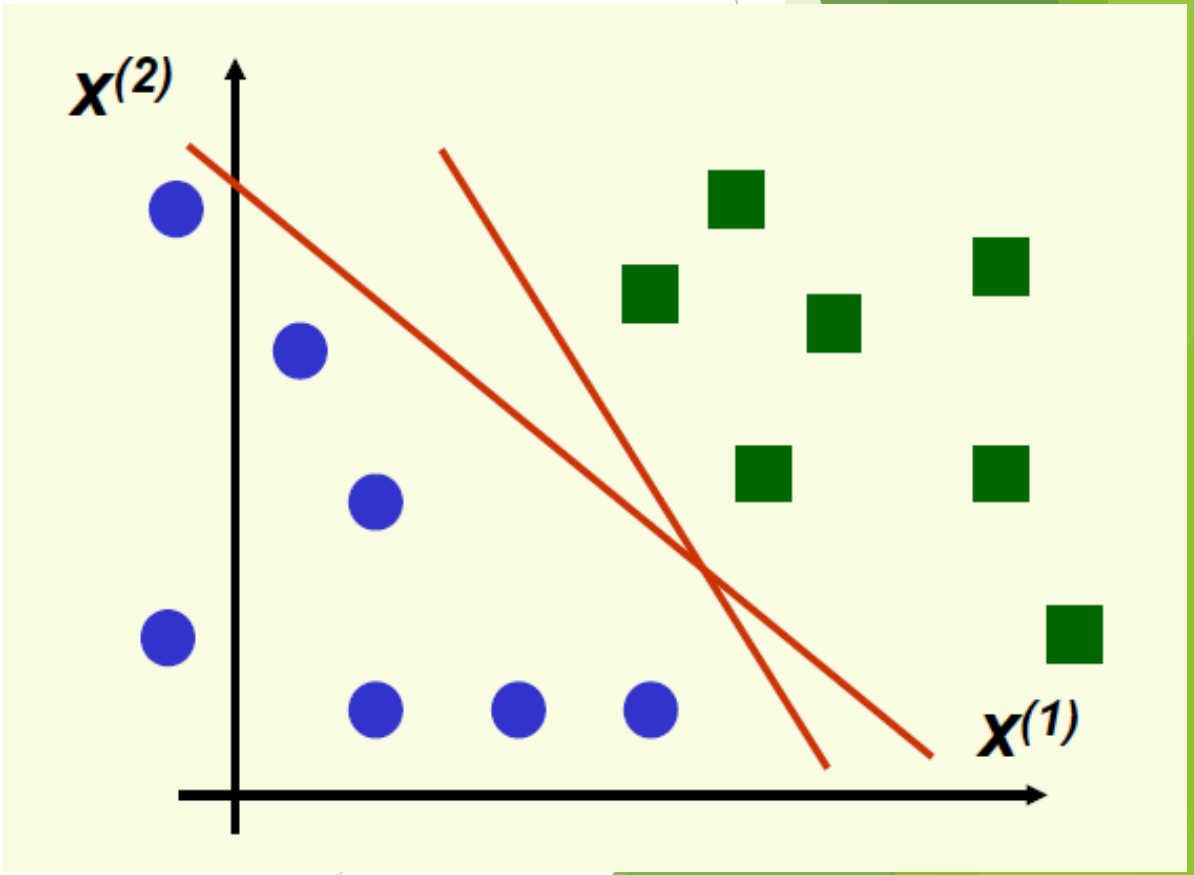
Linear Discriminant Functions

- A discriminant function is linear if it can be written as

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$$

- g(x) > 0 -> x in class 1
- g(x) < 0 -> x in class 2

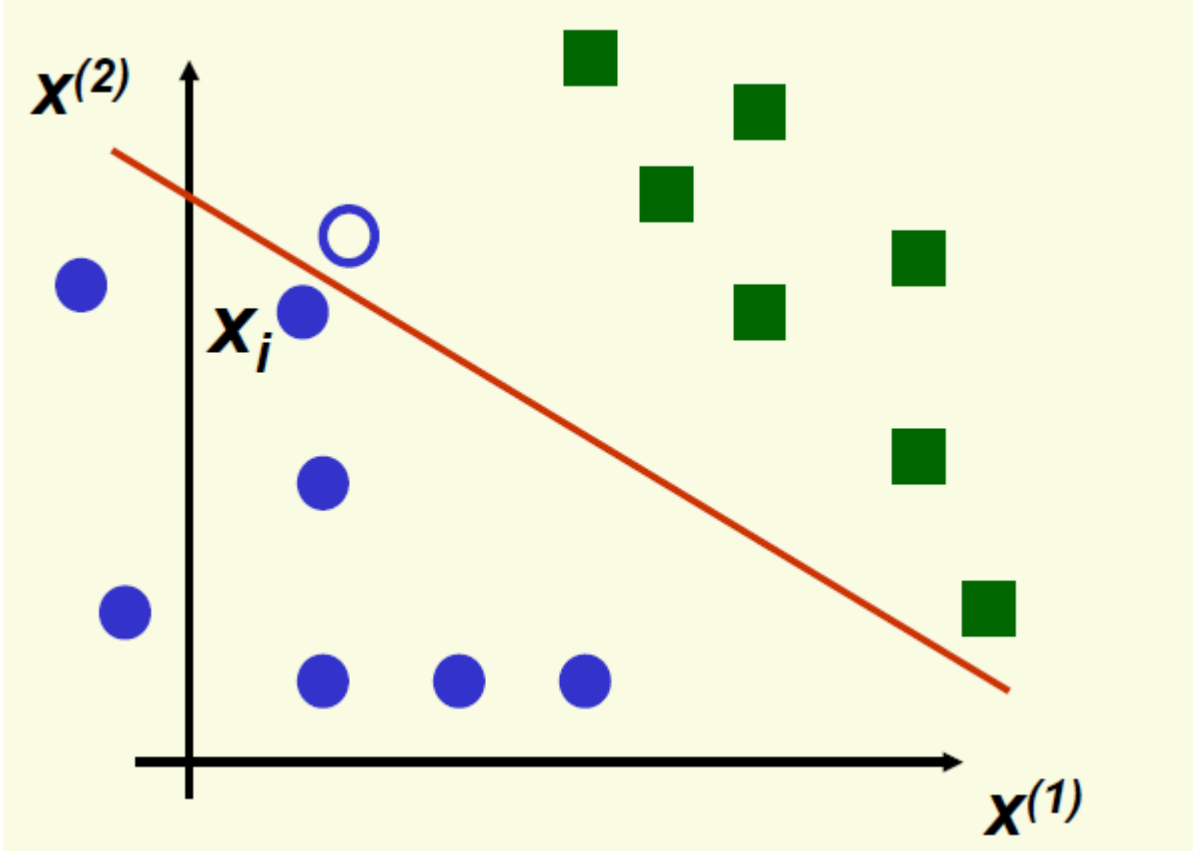
- Which separating hyperplane should we choose?



Linear Discriminant Functions

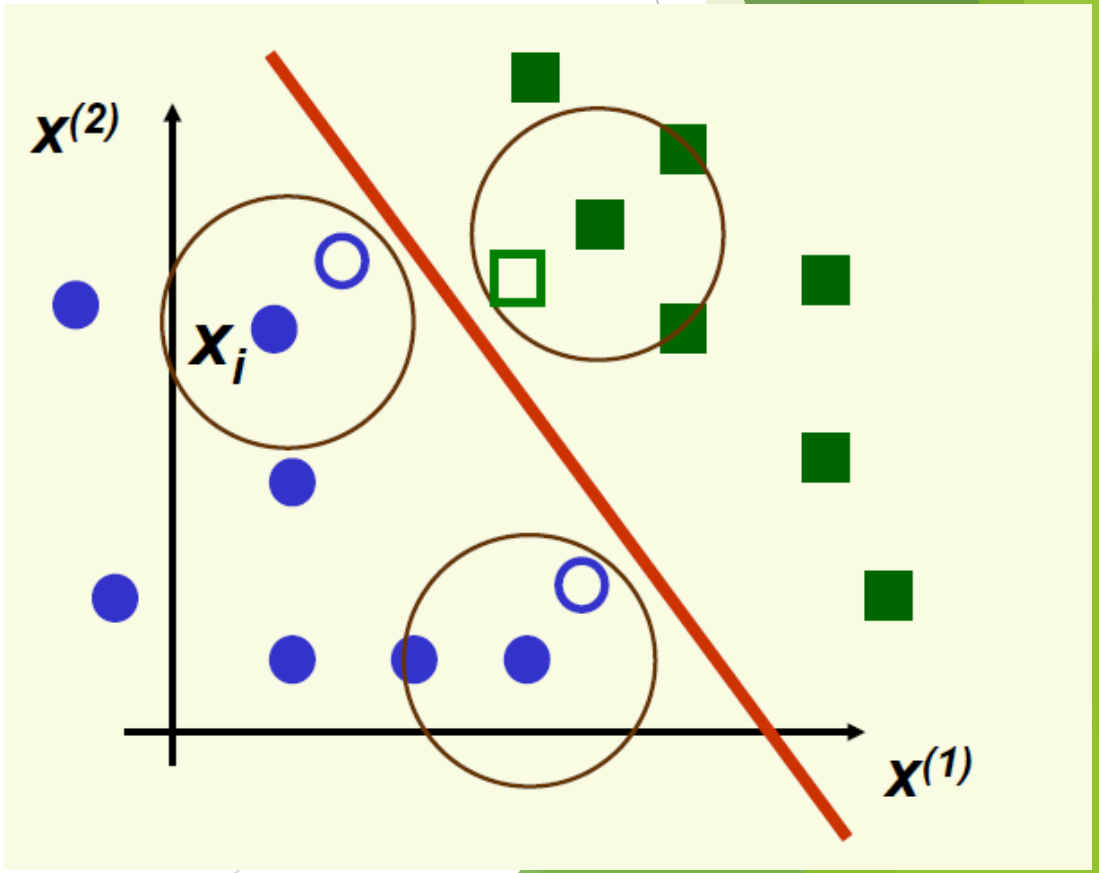
- Training data is just a subset of all possible data
- Suppose hyperplane is close to sample x_i
- If we see new sample close to sample i, it is likely to be on the wrong side of the hyperplane

- Poor generalization (performance on unseen data)



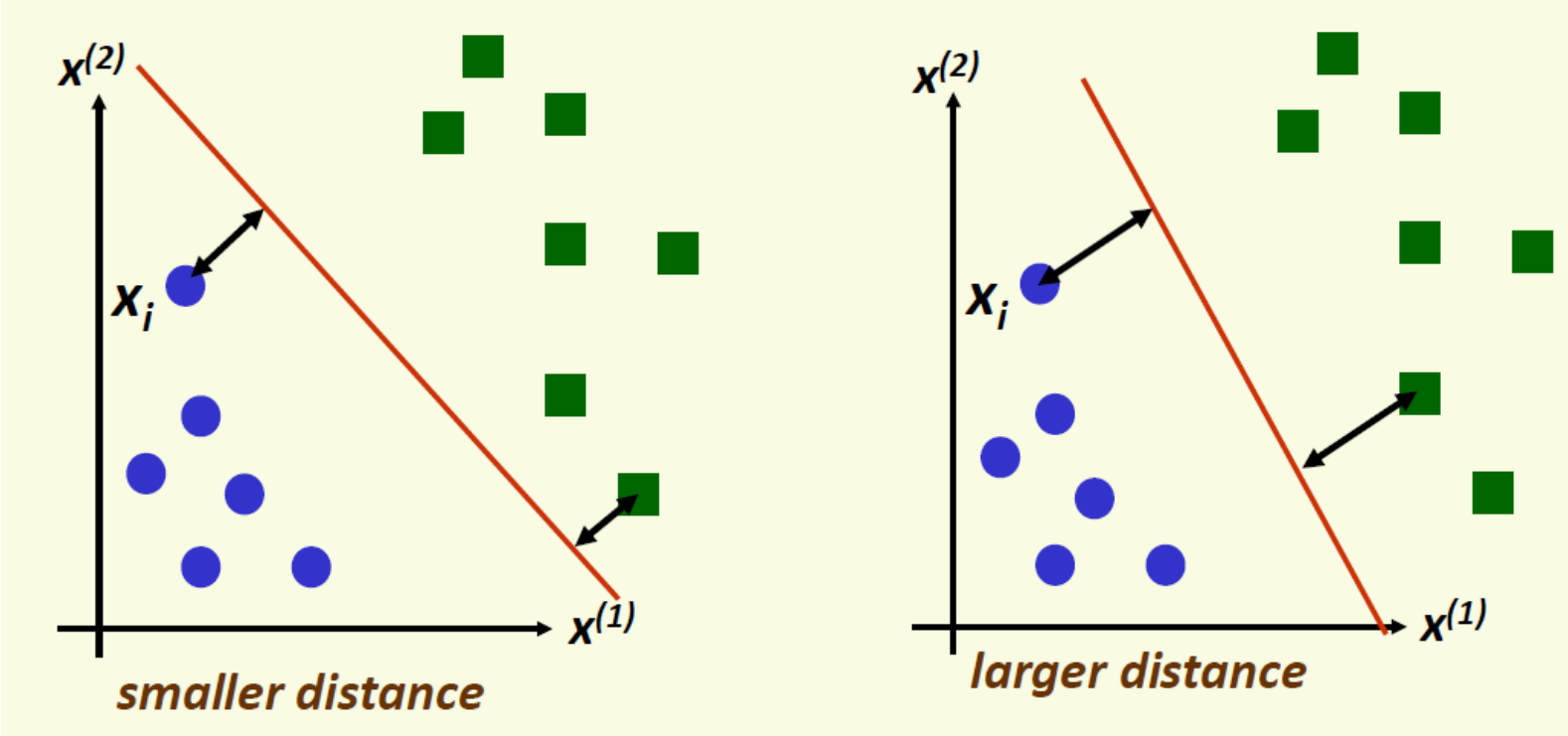
Linear Discriminant Functions

- Hyperplane as far as possible from any sample
- New samples close to old samples will be classified correctly
- Good generalization



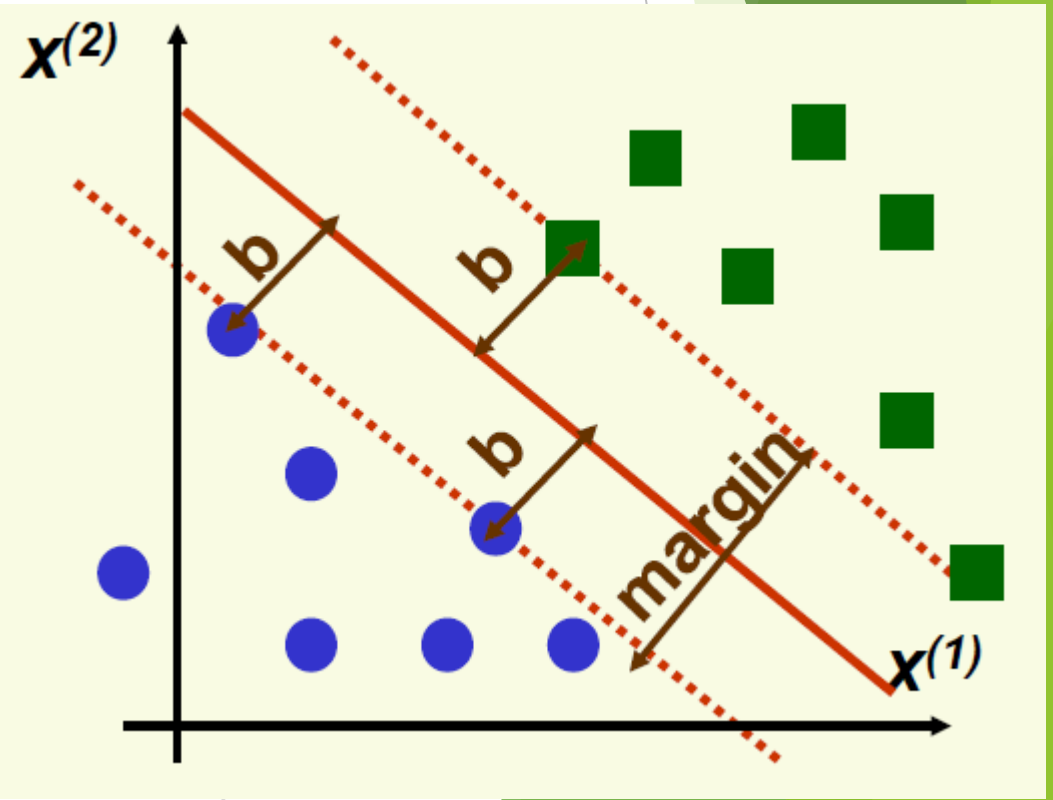
SVM

- Idea: maximize distance to the closest example
- For the optimal hyperplane
  - distance to the closest negative example = distance to the closest positive example



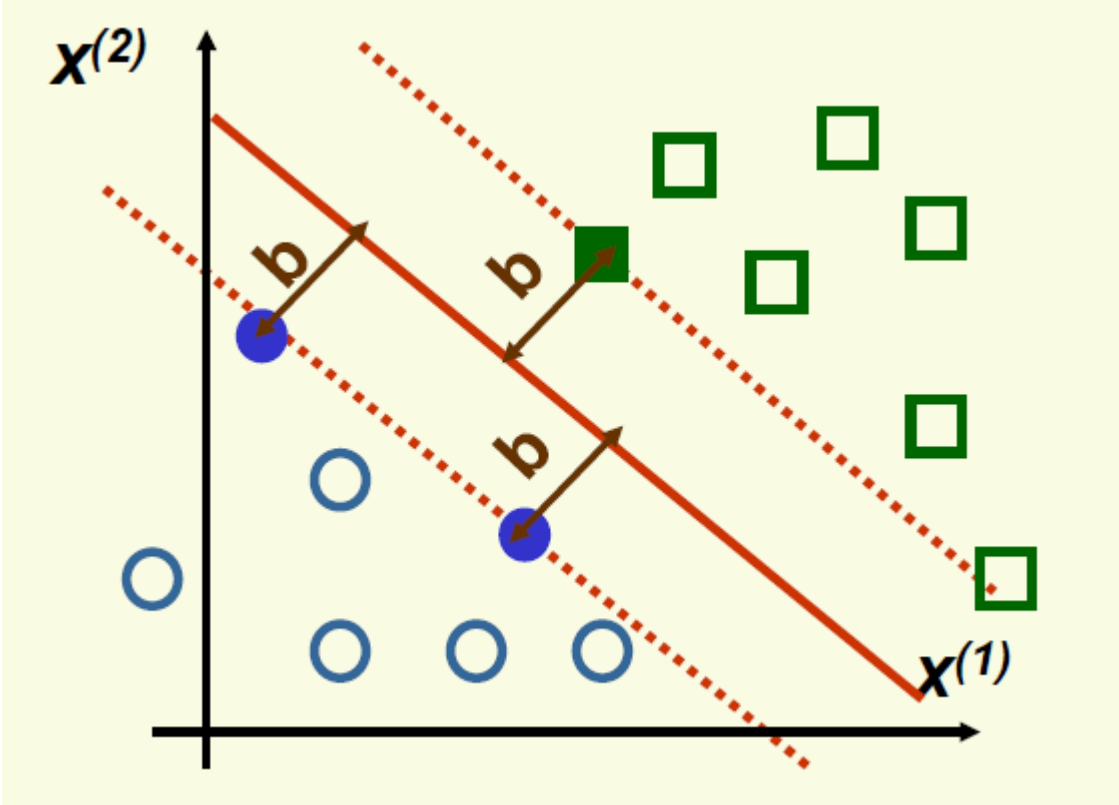
SVM

- SVM: maximize the ***margin***
- ***margin*** is twice the absolute value of distance ***b*** of the closest example to the separating hyperplane
- Better generalization (performance on test data)
  - in practice
  - and in theory



SVM

- ***Support vectors*** are samples closest to separating hyperplane
  - they are the most difficult patterns to classify
  - Optimal hyperplane is completely defined by support vectors
    - (of course, we do not know which samples are support vectors without finding the optimal hyperplane)
- Sounds like a very difficult problem …



SVM

- … but it turns out that it is really not.
- It can be shown that the problem of maximizing the margin can be converted to the following optimization problem

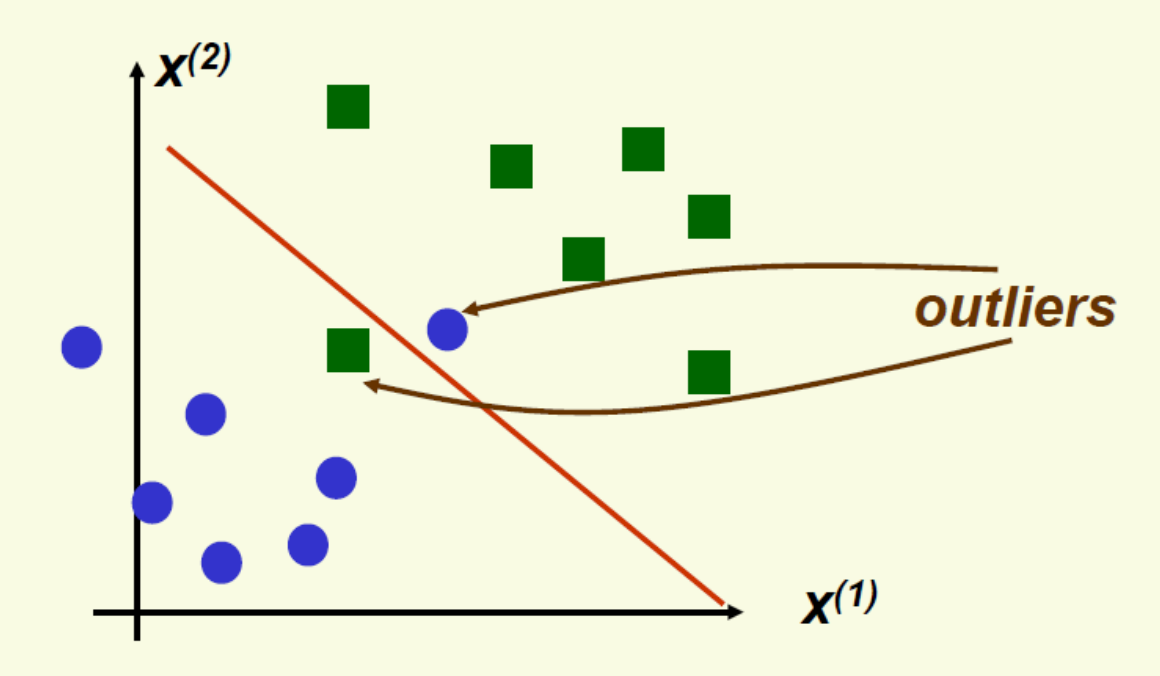
$$\text{minimize} \quad J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$\text{constrained to} \quad z_i\left(\boldsymbol{w}^t \boldsymbol{x}_i + w_0\right) \geq 1 \quad \forall i$$

- Convex problem with a single global minima
- More complex algorithms required (but still efficient)
- Very elegant theory connected to Lagrangian duality

SVM – Non Separable Case

- Data is most likely to be not linearly separable, but linear classifier may still be appropriate



- Can apply SVM in non linearly separable case
  - data should be “almost” linearly separable for good performance

SVM – Non Separable Case

- The original optimization problem is modified in the following way

$$J(w,\xi_1,\ldots,\xi_n)=\frac{1}{2}\|w\|^2+\beta\sum_{i=1}^{n}\xi_i$$

$$\begin{cases} z_i\left(w^t x_i+w_0\right)\geq 1-\xi_i & \forall i \\ \xi_i \geq 0 & \forall i \end{cases}$$

- Still convex problem with a single global minima

SVM Summary

- Advantages:
  - based on nice theory
  - works on non separable data
  - excellent generalization properties
  - objective function has no local minima
  - Complexity of the classifier is characterized by the number of support vectors rather than the dimensionality of the transformed space
- Disadvantages:
  - tends to be slower than other methods
  - quadratic programming is computationally expensive

Classifiers

- **Simple Classifiers**
  - K-Nearest Neighbour (kNN)
  - Naïve Bayes
  - Decision trees
- **Linear Classifiers**
  - Perceptron (Neural Networks)
  - Support Vector Machines (SVM)