Learning

photo: fisher-price.com

Professor Marie Roch

Chapter 19, Russell & Norvig

Learning

- Agents can learn to improve:
  - inference from percepts
  - information about world evolution
    - as the result of a changing world
    - as the result of actions
  - utility estimators
  - action choices
    - either update condition-action maps
    - goal modification to maximize utility

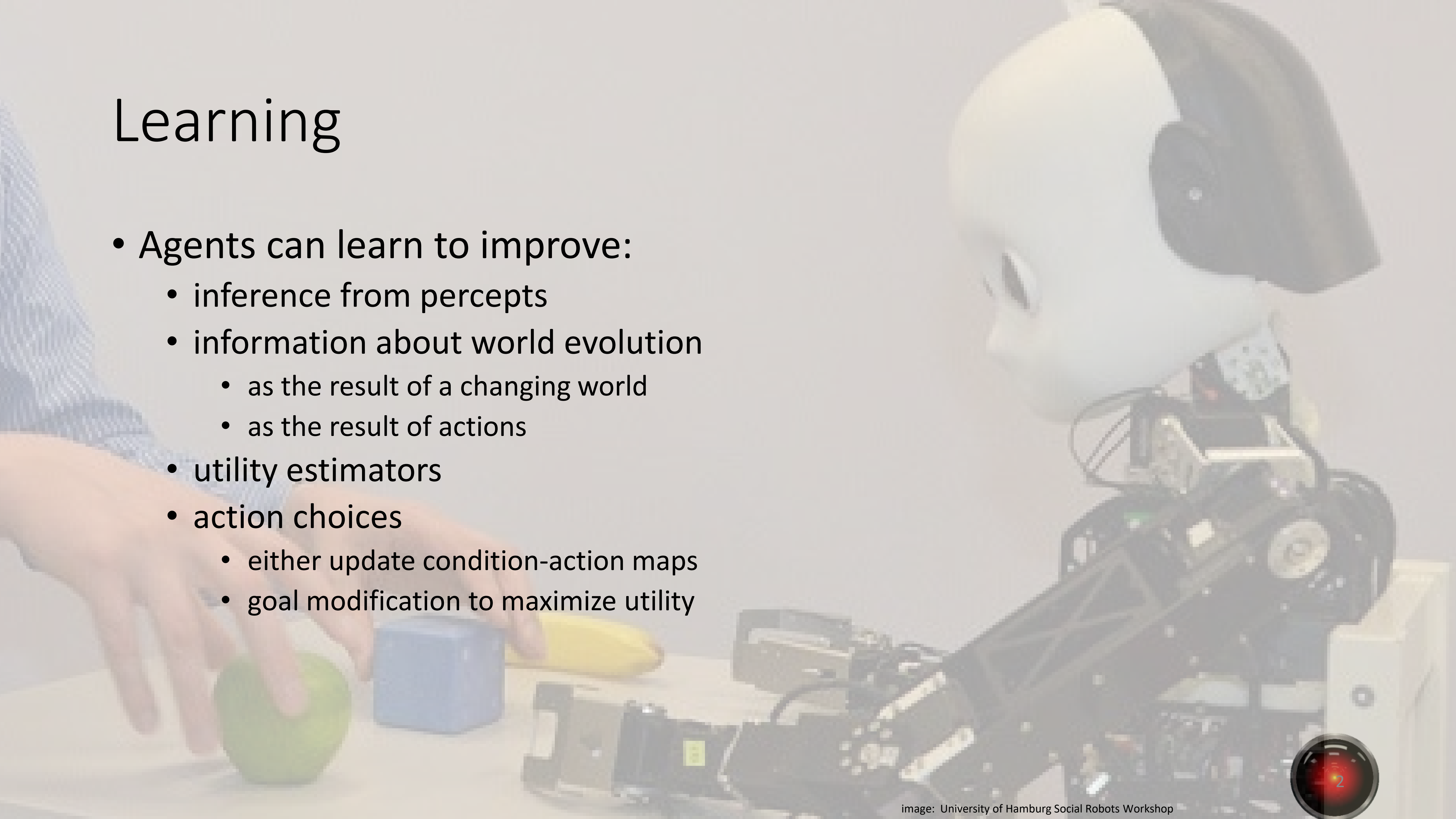


image: University of Hamburg Social Robots Workshop

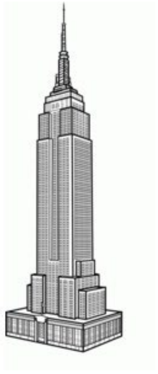
What we want to learn

- Mapping function
    - Inputs are factored representations
      e.g. a vector of values
    - Outputs are
        - discrete (e.g. categorical)
        - continuous

Types of learning

- Inductive – Learn map between between input/output pairs

ice cream



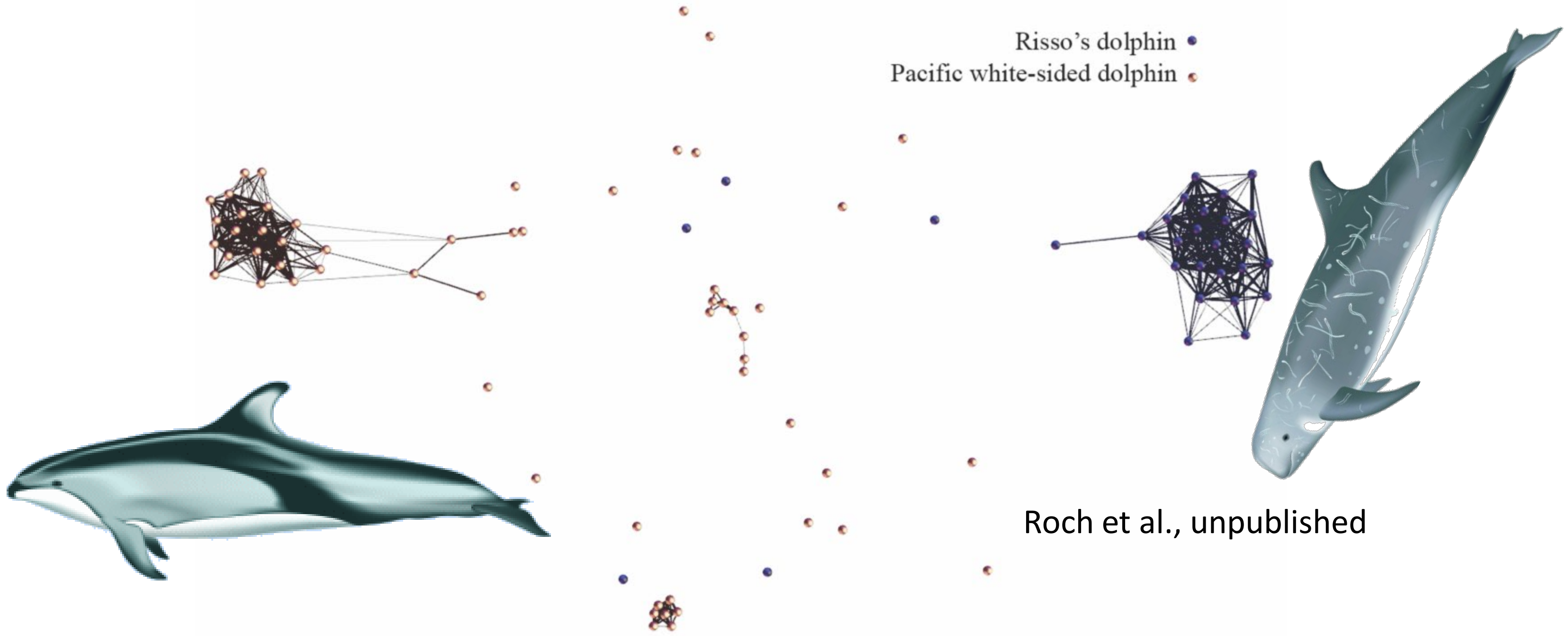
building

horse

- Deductive – Creating rules that are logically entailed, such as if I am in a dark cave and I don’t feel a breeze, I’m not going to step into a pit.

Learners vary based on their feedback

- Unsupervised learning
  - No explicit feedback
  - Goal is to cluster "similar" things



Roch et al., unpublished

Learners

- Reinforcement learning
  - Learner is given rewards/punishments for actions
  - Example: Positive reinforcement animal training

- Supervised learning
  - Each input is paired with a label or value and the agent attempts to learn to predict the labels/values for novel data.

- Hybrids are possible, such as semi-supervised learning where a small set of labeled data accompanies a large set of unlabeled data.

Caveat about labeled data sets

- We refer to labels as “ground truth.”

- One should be cautious with ground truth…
  Why?

Supervised learning

- Suppose there exists an unknown f: x → y such that

$$\left(y_1 = f(x_1)\right) \wedge \left(y_2 = f(x_2)\right) \wedge \left(y_3 = f(x_3)\right) \wedge \ldots$$

  and we are given only a *training set*

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$$

- Supervised learning estimates a function h: x→y that approximates f.

Supervised learning

- Function h is the *hypothesis* and our estimation is a search in *hypothesis function space* for a good hypothesis

- Learning is a search for a good hypothesis.
- How do we measure goodness?
  - Evaluate the function on a labeled *test* set.
  - The test set must be distinct from the training set:

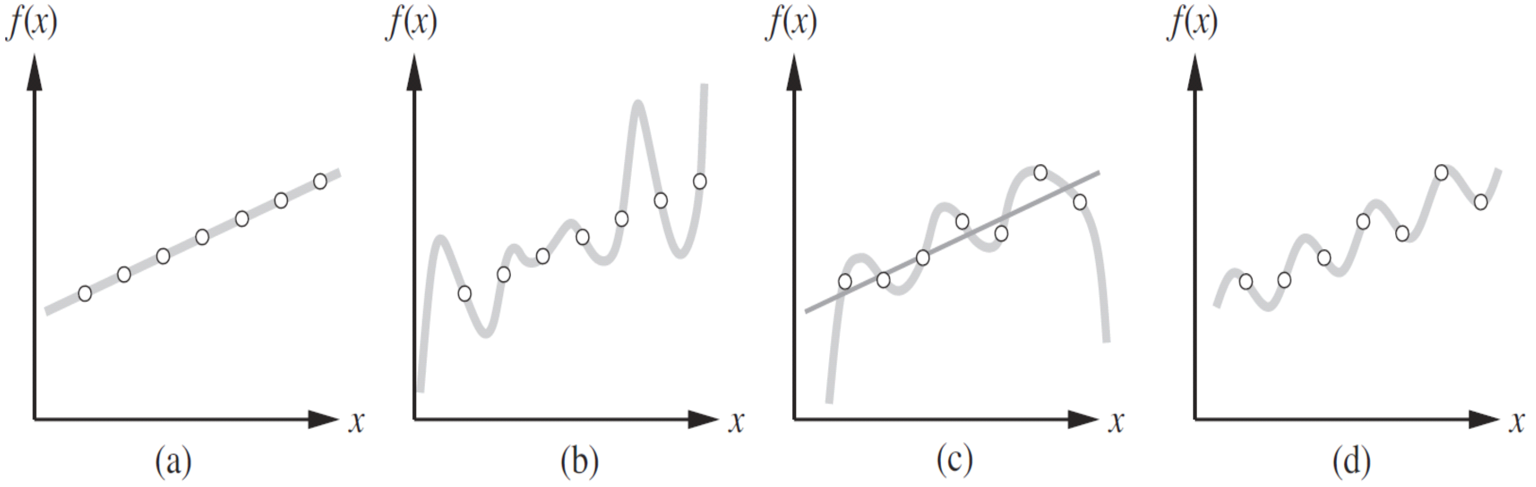
    $$\text{training} \cap \text{test}=\emptyset$$

  - We say h *generalizes* well if it performs well on the test set.

Why do we need to test on novel data?

How to choose amongst functions?

Ockham's razor – Use simplest hypothesis consistent with the data



William of Ockham 1287-1347

All of these functions fit the training data, but which one is most likely to correctly predict new data?

Hypothesis spaces

- The more complex a hypothesis space,
  the more difficult it is to find a good hypothesis.

- Fits well with Ockham’s Razer.

Experience data set

- We learn from training data
- Can be organized into a design matrix,
  a set of experiences:

$$\begin{array}{cccccc} & & \text{feature vector} & & & \text{label (supervised problems)} \\ \left[\begin{matrix} x_{1,1} \\ x_{2,1} \\ x_{3,1} \\ \vdots \\ x_{N,1} \end{matrix}\right. & \begin{matrix} x_{1,2} \\ \cdots \\ \cdots \\ \\ x_{N,2} \end{matrix} & \begin{matrix} x_{1,3} \\ \\ \\ \\ x_{N,3} \end{matrix} & \begin{matrix} \cdots \\ \\ \\ \\ \cdots \end{matrix} & \begin{matrix} x_{1,D} \\ \\ \\ \\ x_{N,D} \end{matrix} & \left.\begin{matrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{matrix}\right] \end{array}$$

Learning sets of functions may be used if some features are missing.
Example functions:
  f_0 for all features,
  f_1 if feature 1 is missing, etc.
Can also attempt to fill in missing data

Supervised learning

- Mapping f could be stochastic
  - If so, f is not a function of x
  - In these cases, we learn a conditional probability distribution P(Y|x)
- What are we learning:
  - y is categorical → *classification*
    - example: $y \in$ {happy, sad, angry, serious}
    - binary classifier – special case with exactly two classes
  - y is numeric → *regression*
    - example: y = change in sea level (m) since 1990

Regression

- Fit a function to experience data
- We start with linear regression and a family of functions on input feature vector x:

$$\sum_{i=1}^{D} w_i x_i = \hat{y} \text{ or in matrix notation } w^T x = \hat{y}$$

$$[w_1 \; w_2 \; w_3 \ldots w_D] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_D \end{bmatrix} = \hat{y}$$

- Goal: learn *w* such that $w^T x = \hat{y} \approx y$

19.6 in Russell & Norvig

Regression

- Loss functions measure performance
  Example: Squared error loss $L(y, \hat{y}) = (y - \hat{y})^2$
- Mean squared loss (MSL) is the average squared loss
- The normal equation is a closed form solution to select the w that minimizes MSL given a design matrix

$$w = (X^T X)^{-1} y^T X$$

$$\left[\underbrace{\begin{matrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,D} \\ x_{2,1} & \cdots & & & \\ x_{3,1} & \cdots & & & \\ \vdots & & & & \\ x_{N,1} & x_{N,2} & x_{N,3} & \cdots & x_{N,D} \end{matrix}}_{X} \quad \overbrace{\begin{matrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{matrix}}^{y}\right]$$

- Interesting, but we will look at this differently

Gradient descent regression

- Suppose we want to minimize loss for a design matrix
- We could compute the gradient with respect to the weights w

$$\nabla_w L(y, \hat{y}) = \nabla_w (y - \hat{y})^2 = \nabla_w (y - w^T x)^2$$
$$\text{as } \hat{y} = w^T x$$

- This is a vector that indicates the direction in which loss increases the fastest

Gradient descent regression

- Concrete example:
- Row from design matrix: [2 1 12]

$$w = \begin{bmatrix}3\\4\end{bmatrix}, x = \begin{bmatrix}2\\1\end{bmatrix} \rightarrow \hat{y} = w^T x = [3 \quad 4]\begin{bmatrix}2\\1\end{bmatrix} = 10$$

$$\mathrm{L}(\mathrm{y}, \hat{y}) = (y - \hat{y})^2 = (12 - 10)^2 = 4$$

$$\nabla_w L(y, \hat{y}) = \begin{bmatrix} \frac{\partial}{\partial w_1}(12 - (2w_1{+}1w_2))^2 \\ \frac{\partial}{\partial w_2}(12 - (2w_1 + 1w_2))^2 \end{bmatrix}$$

Gradient descent regression

$w = \begin{bmatrix}3\\4\end{bmatrix}, x = \begin{bmatrix}2\\1\end{bmatrix}$. Remember $\frac{d}{du}u^p = pu^{p-1}du$

$$\nabla_w L(y,\hat{y}) = \begin{bmatrix} 2(12-(2w_1+1w_2))\frac{\partial}{\partial w_1}(12-(2w_1+1w_2)) \\ 2(12-(2w_1+1w_2))\frac{\partial}{\partial w_2}(12-(2w_1+1w_2)) \end{bmatrix}$$

$$= \begin{bmatrix} 2(12-(2w_1+1w_2))(-2w_1) \\ 2(12-(2w_1+1w_2))(-w_2) \end{bmatrix}$$

$$= \begin{bmatrix} 2\big(12-(6+4)\big)(-2\cdot 3) \\ 2\big(12-(6+4)\big)(-1\cdot 4) \end{bmatrix} = \begin{bmatrix} 2\cdot 2\,(-2\cdot 3) \\ 2\cdot 2(-1\cdot 4) \end{bmatrix} = \begin{bmatrix} -24 \\ -16 \end{bmatrix}$$

Moving in this direction increases loss fastest

Gradient descent regression

- Adapting weights:

$$w_{new} = w - \alpha \nabla_w L(y, \hat{y})$$

- Why do we subtract?
- α is the learning rate
  some authors use other letters, e.g. ϵ

- Adapting the weights for each sample results in wildly different gradient directions

Batch gradient descent algorithm

```
initialize w
while not done:
    gradient = 0
    for x, y in design matrix:
        gradient += $\nabla_w L(w^T x, y)$
    w = w – alpha * gradient
    done = meets criterion?
      e.g., $\nabla_w L(\cdot)$ plateaus or max# iterations
```

Stochastic gradient descent

- Batch gradient descent is slow

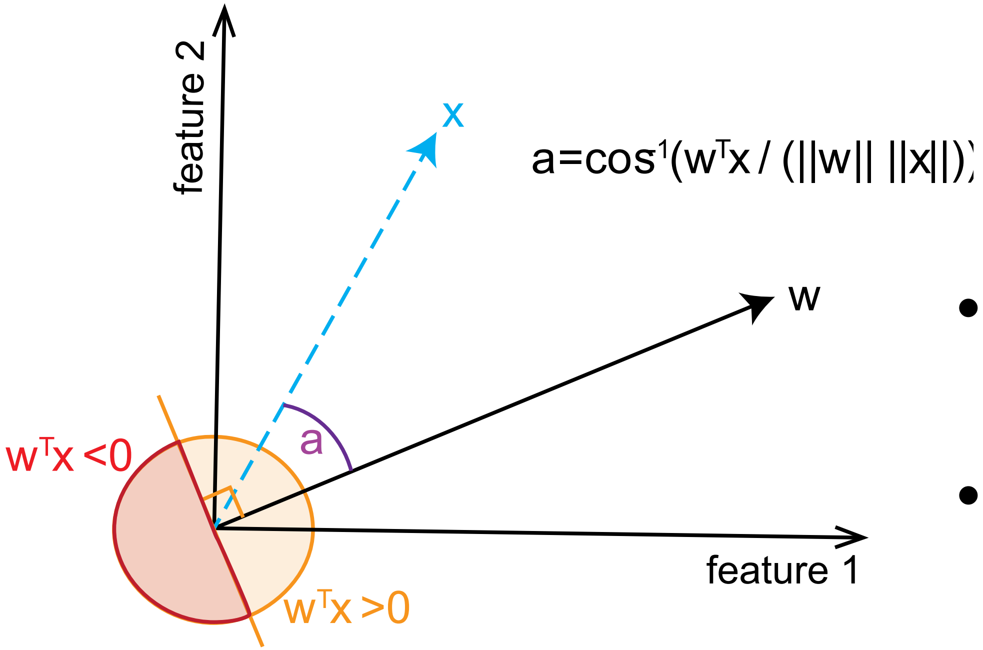
- Random minibatches speed things up
  - Randomly batch examples into minibatch groups of N
  - Update weights based on minibatch
  - Generally converges to a solution faster

Why might it be important to randomize the minibatches?

Regression based classification

- Suppose our labels are -1 and 1.
- Design matrix now specifies a binary classification problem
- We can use the same techniques to learn *w*

Interpreting weight vectors



Roch et al. 2021, *Acoustics Today*

- $w^T x \propto \angle a$
  (note: $w^T x$ =||w|| · ||x|| ·cos(a))
- Sign indicates which side of line ⊥ to w vector x falls on

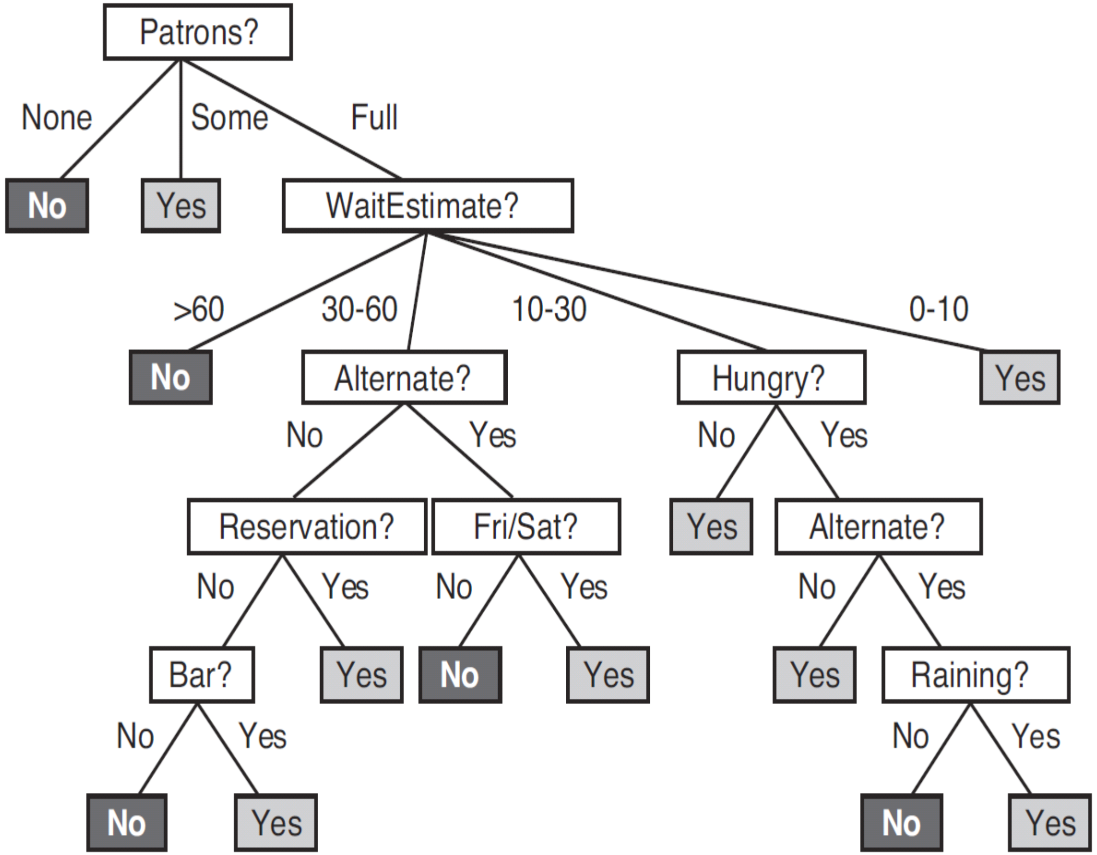
drawing: MontyPython, print Madame Bricolage Press

Decision tree learner

- Answers a series of questions to arrive at a solution

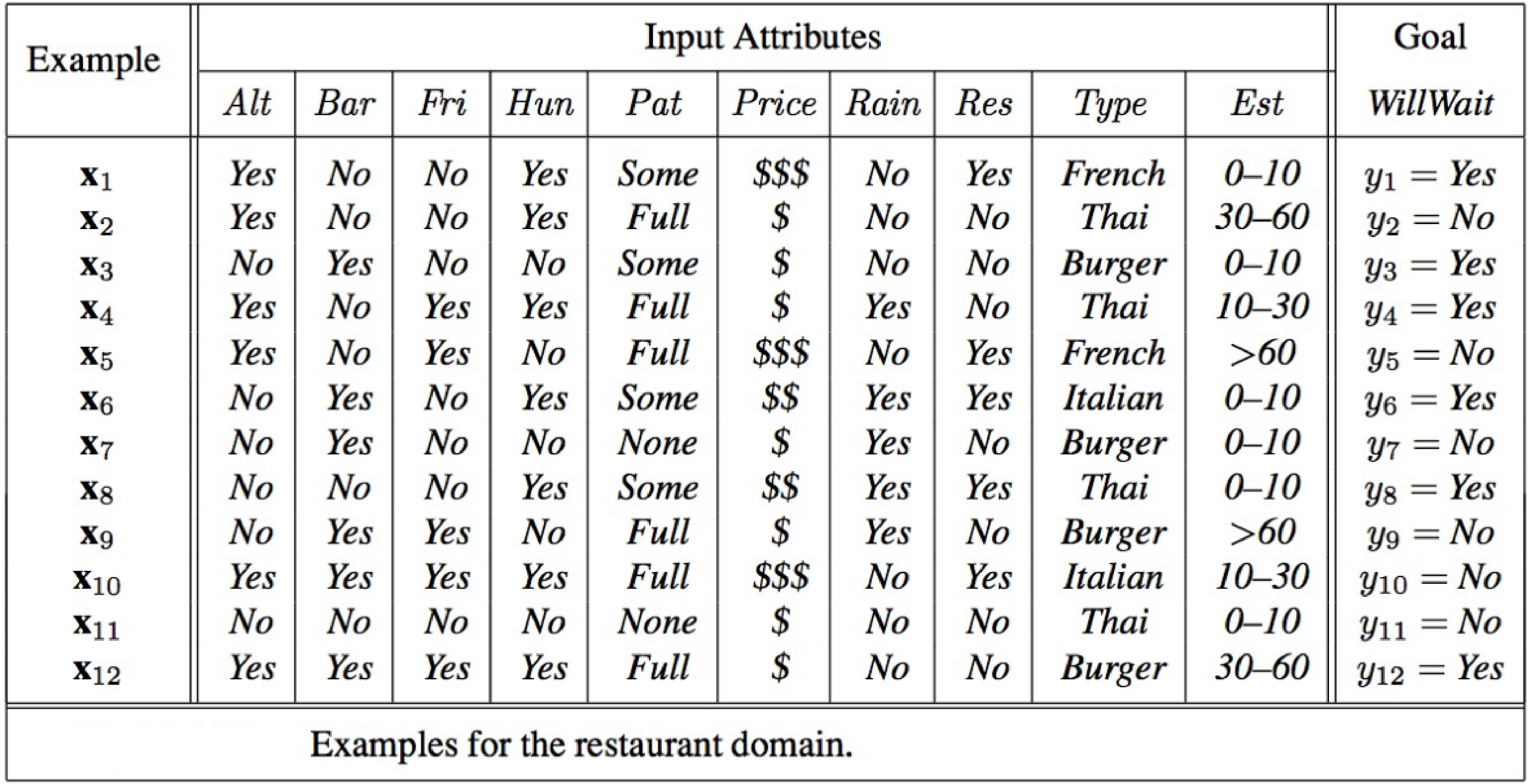
- For now, we restrict our discussion to
  - questions that have categorical (discrete) answers
  - binary classification decisions

Dr. Stuart Russell is hungry…



Professor Russell's decision tree for where to eat…
9 questions from 10 attributes (price is not used)

Learning a tree from examples



Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Examples for the restaurant domain.

Figure 19.2 R&N p. 657

Constructing a tree from examples

- Which question to ask first?

- What do you look for when you play 20 questions?

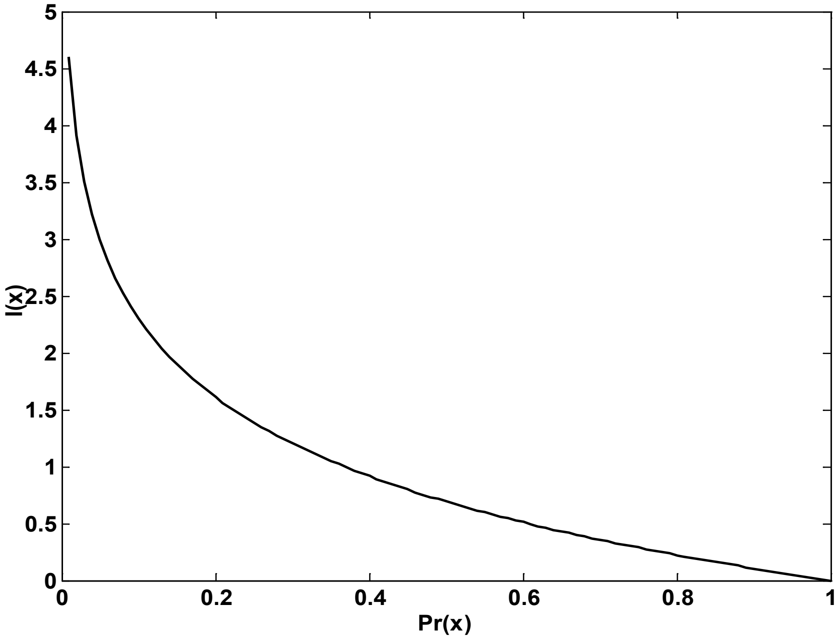
  Chances are, you intuitively use information theory…

Quantity of information

- Amount of surprise that one sees when observing an event

- We obtain a large quantity of information (measured in bits) from rare events

$$I(x_i) = \log_2 \frac{1}{\mathrm{P}(x_i)}$$



Note: We use log base 2 and will start omitting the base later on.

Expectation

- An expected value is the value that we expect to see most often.
- We sum the product of each possible value and the probability that it occurs

$$E[X] = \sum_{x_i \in S} x_j \, \mathrm{P}(\, x_i) \text{ where S is the set of all possible values of X}$$

- Example
  - Pick a number between 1-10 with
    - all numbers except 7 equally likely.
    - 7 is three times more likely to be picked

$$P(X = x) = \begin{cases} \frac{1}{12} & x \neq 7 \\ \frac{3}{12} & x = 7 \end{cases} \text{, so } \mathrm{E[X]} = \textstyle\sum_{i \neq 7} i \, \frac{1}{12} + 7 \, \frac{3}{12} \; = 5.75$$

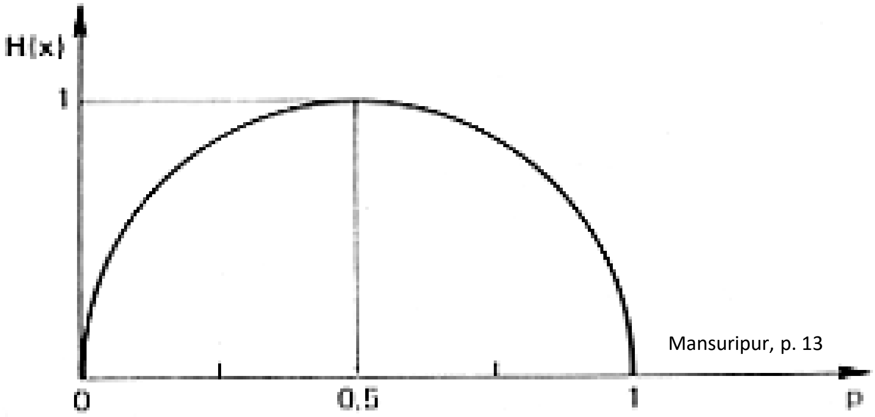
Entropy

- Entropy is defined as the expected amount of information (average amount of surprise) and is usually denoted by the symbol H.

$$
\begin{aligned}
H(X) &= E[I(X)] \\
&= \sum_{x_i \in S} \mathrm{P}(x_i) I(x_i) && S \text{ is all possible symbols} \\
&= \sum_{x_i \in S} \mathrm{P}(x_i) \log_2 \frac{1}{\mathrm{P}(x_i)} && \text{definition } I(x_i) \\
&= E[-\log_2 \mathrm{P}(X)]
\end{aligned}
$$

Example

- Assume
  - X = {0, 1}
  - $\mathrm{P}(X) = \begin{cases} p & X = 0 \\ 1-p & X = 1 \end{cases}$



Mansuripur, p. 13

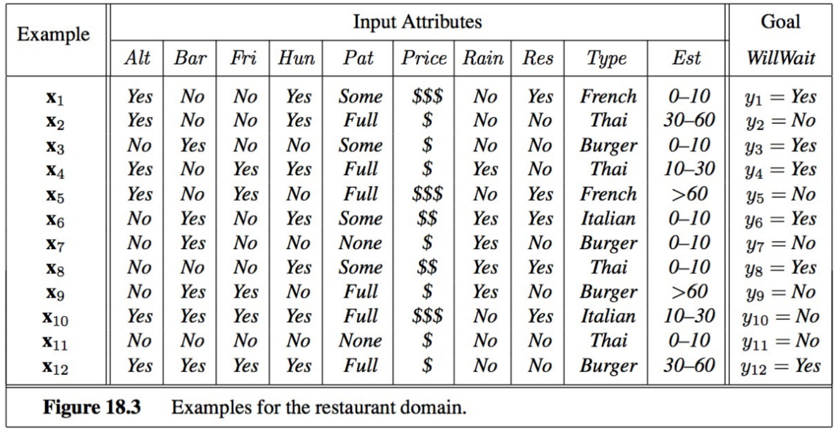
H(x) versus p

- Then

$$\begin{aligned} H(X) &= E[I(X)] \\ &= -p\log p - (1-p)\log(1-p) \end{aligned}$$

Restaurant example

- WillWait response:
  - 6 positive
  - 6 negative



Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

- Entropy

$$
\begin{aligned}
H(x) &= -\frac{p}{p+n}\log_2\frac{p}{p+n} + -\frac{n}{p+n}\log_2\frac{n}{p+n} \\
&= -\frac{6}{6+6}\log_2\frac{6}{6+6} + -\frac{6}{6+6}\log_2\frac{6}{6+6} \\
&= \log_2 2 = 1
\end{aligned}
$$

Entropy and tree questions

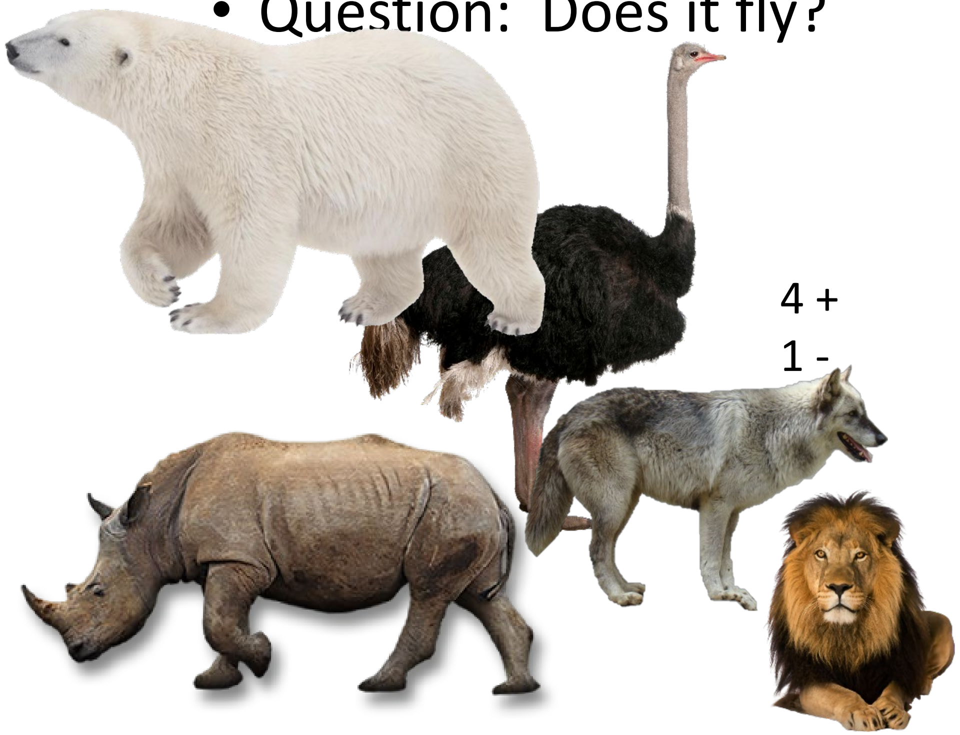
- Fig. 18.3 has an equal number of positive and negative examples (6 each: p=n=6)
- Training data has entropy of 1 bit:

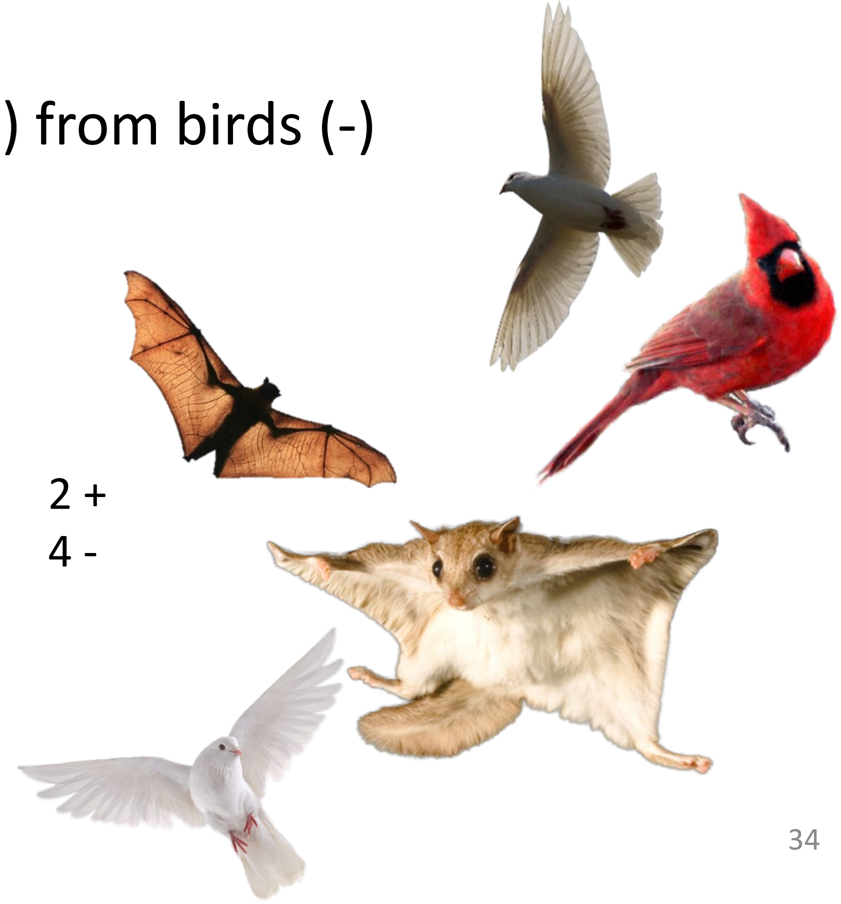
$$
\begin{aligned}
H(x) &= -\frac{p}{p+n}\log_2\frac{p}{p+n} + -\frac{n}{p+n}\log_2\frac{n}{p+n} \\
&= -\frac{1}{2}\log_2\frac{1}{2} + -\frac{1}{2}\log_2\frac{1}{2} \\
&= \log_2 2 = 1
\end{aligned}
$$

Tree questions

Animal set:
6 mammals (+), 5 birds (-)

- Tree questions have a binary response
- Suppose
  - Goal: separate mammals (+) from birds (-)
  - Question: Does it fly?





Tree question entropy

- Remember, entropy is: $E\left[I\big(P(X)\big)\right] = E\left[-\log_2 \mathrm{P}(X)\right]$
- For binary categories, we define a short hand:
  - $q = {}^{p}/_{p+n}$, , the positive rate
  - $1 - q = {}^{n}/_{p+n}$, the negative rate
  - $B(q) = \mathrm{E}[\mathrm{I}\big(\mathrm{P}(\mathrm{X})\big)] = -q\log_2 q - (1-q)\log_2(1-q\big)$

Tree question entropy

Bird/mammal example

	p	n	q (+ rate)	B(q)
before question	6	5	6/11	0.99
¬flies	4	1	4/5	0.72
flies	2	4	1/3	0.92

Sample computation flies:

$$q = \frac{2}{2+4} = \frac{1}{3}$$

$$B\left(\frac{1}{3}\right) = -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \left(1-\frac{1}{3}\right)\log_2\left(1-\frac{1}{3}\right) = \frac{1}{3}\log_2\left(\frac{1}{3}\right) - \left(\frac{2}{3}\right)\log_2\left(\frac{2}{3}\right) \approx .92$$

Entropy and tree questions

- Patrons – Categories (None, Some, Full)

  None: 2 examples: B(0/2) = 0

  Some: 4 examples: B(4/4) = 0

  Many: 6 examples: B(2/6) = .918

- Restaurant type (French, Italian, Thai, Burger)

  French: B(1/2) = 1

  Italian: B(1/2) = 1

  Thai: B(2/4) = 1

  Burger: B(2/4) = 1

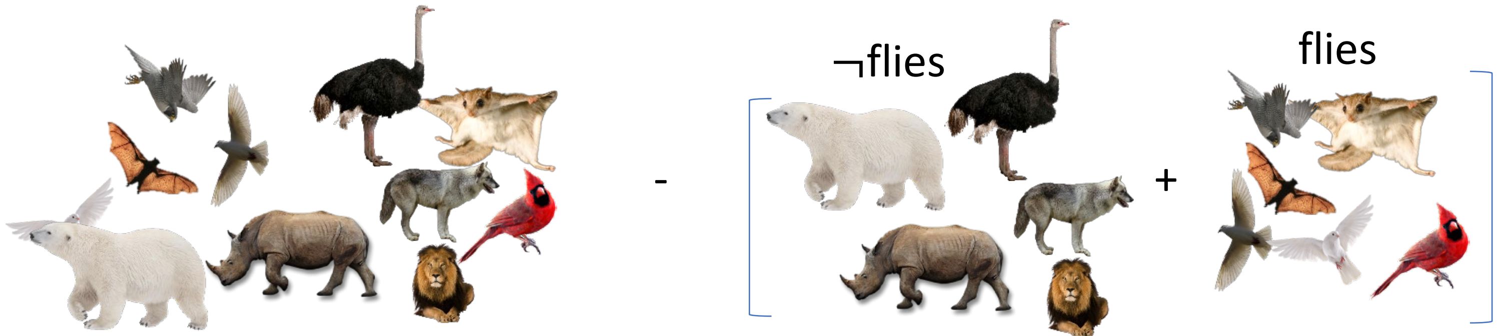
Information gain

- Goal: reduce the amount of information needed to represent the problem
- We can represent the remaining entropy after dividing data into d groups with question A as follows:

$$\text{Remainder}(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right)$$

and the information gain as:

$$Gain(A) = B\left(\frac{p}{p + n}\right) - \text{Remainder}(A)$$



Information gain examples

- Mammal/bird flight question
  - Split 11 animals into two groups of size 5 (¬flies) and 6 (flies).

  - $\text{Remainder(Does it fly?)} = \underbrace{\frac{2+4}{6+5} B\left(\frac{1}{3}\right)}_{flies} + \underbrace{\frac{4+1}{6+5} B\left(\frac{4}{5}\right)}_{\neg flies} = \frac{6}{11} \cdot .92 + \frac{5}{11} \cdot .72 \approx .83$

  - $\text{Gain(Does it fly?)} = B\left(\frac{6}{6+5}\right) - \text{Remainder(Does it fly?)}$
    $= 0.99 - 0.83 = 0.16$

Information gain examples

- Patrons – Categories (None, Some, Full)

  None: 2 examples: B(0/2) = 0
  Some: 4 examples: B(4/4) = 0
  Many: 6 examples: B(2/6) = .918

$$Gain(Patrons) = B\left(\frac{6}{6+6}\right) - \left(\frac{2}{12}\cdot 0 + \frac{4}{12}\cdot 0 + \frac{6}{12}\cdot .918\right) \approx .541 \text{ bits}$$

- Restaurant type (French, Italian, Thai, Burger)

  French: B(1/2) = 1
  Italian: B(1/2) = 1
  Thai: B(2/4) = 1
  Burger: B(2/4) = 1

$$Gain(Type) = B\left(\frac{6}{6+6}\right) - \left(\frac{2}{12}\cdot 1 + \frac{2}{12}\cdot 1 + \frac{4}{12}\cdot 1 + \frac{4}{12}\cdot 1\right) = 0 \text{ bits}$$

Decision tree learner

```
def decision-tree-learner(examples, attributes, parent_examples):
    if empty(examples):
        return plurality-value(parent_examples)  # pick whatever parent had most of
    else if all examples of same class:
        return the class
    else if empty(attributes):  # no more questions to ask
        return plurality-value(examples)
    else:
        a = arg max$_{a \in attributes}$ importance(a)      # information gain or other measure
        t = new tree(a)       # Create a new tree rooted on most important question
        for each value v associated with attribute a:
            vexamples = {e : e $\in$examples such that e has value v for attribute a}
            subtree = decision-tree-learner(vexamples, attributes – a, examples)
            t.add_branch(v, subtree)  # Add in new subtree with current value as branch label
        return t
```

Will Indie survive?

image credit: Indiana Jones © Lucasfilm Ltd.

Will Indie survive?

- We can build a classifier that predicts if Indiana Jones survives (well, of course he does)
- Possible features:
  - Number of bad guys
  - any snakes?
  - length of Indie's whip
- Some features might not have much to do with survival:
  - Does Indie have his hat?
  - Did Indie brush his teeth?

AbsurdWordPreferred - DeviantArt

Features and overlearning

- Useless features are not good for prediction, but…

  a learner may pick up on random patterns in the training data and incorporate these into the rules

- Example:
  - Task: Random six-sided fair die, learn whether or not we roll 5.
  - Will height from which we roll have any bearing on P(X=5)=.2
  - Decision tree may again pick up random patterns, but the lowest classification error rule is to simply say: we will not roll a 5.

Generalization and overfitting

- Learning random patterns that do not affect the actual function f is called overfitting.
- Overfit models do a great job predicting *training* data, but *do not predict novel data well.*
- Decision trees have a tendency to overfit.

Pruning Decision trees

- Overfitting of decision trees is addressed by pruning.

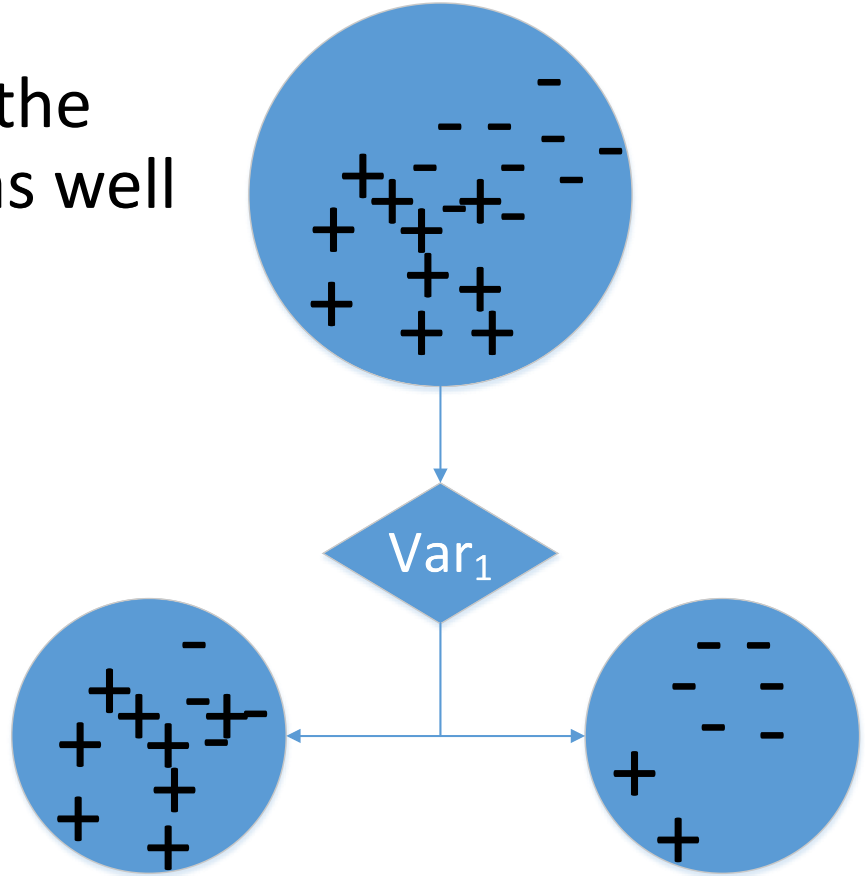
- For each leaf node, we ask ourselves if we had good information gain.
  If the node was informative, we keep it.
  If we didn’t learn anything, we discard.

- NOTE: This is done *after* the tree is trained.

Pruning decision trees

- How do we know if our decisions were any good?

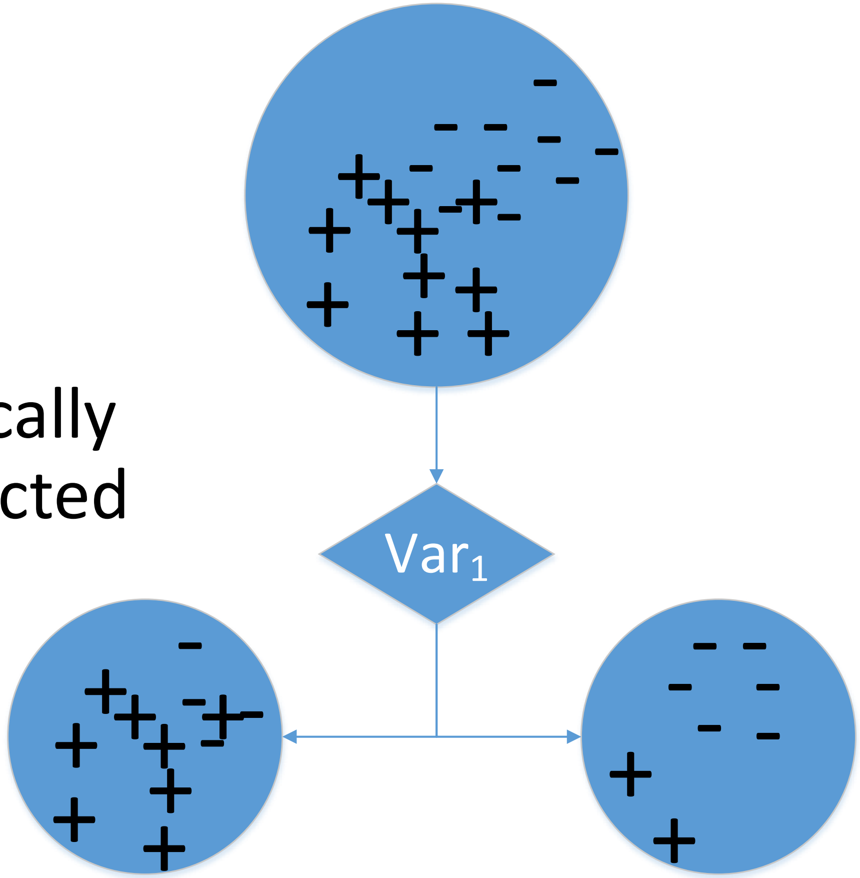
- Our goal was to separate into the positive and negative classes as well as possible.



Pruning decision trees

- Here we did not do a great job of separating.

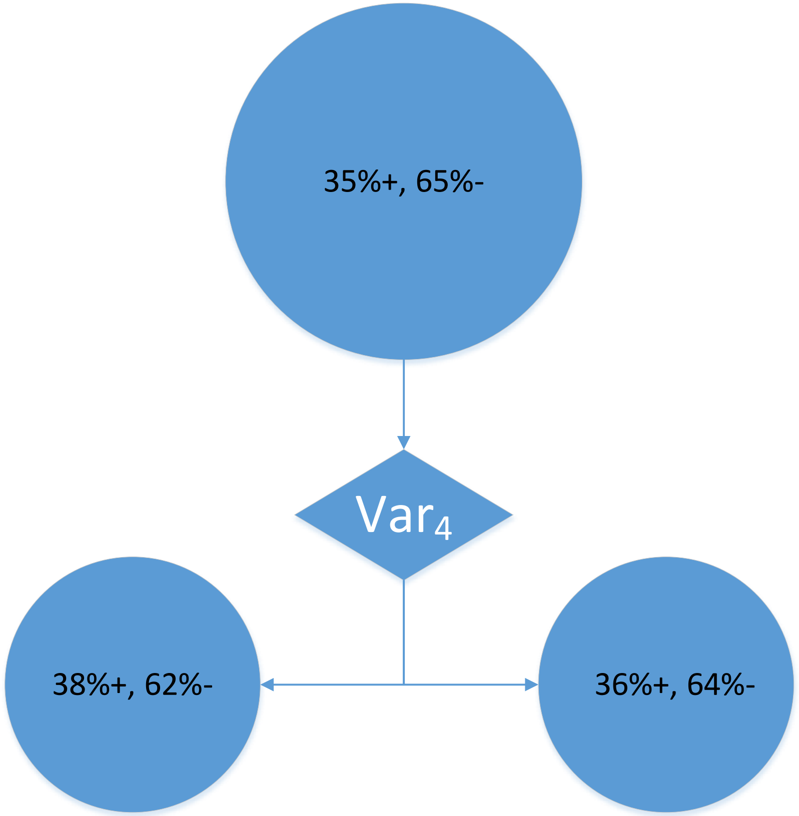
- Can we devise a statistic that lets us know if our observed split is statistically *significantly different* from the expected ratio?



χ^2 Test

- Suppose decision tree splits a node into *v* sets.

- If the node does not add any new information, then we expect each child to have about the same distribution of class labels



χ^2 Test

- Let us restrict an example to our two-class problem with *v*=2 categories.

$$P(p) = \frac{p}{p+n}, P(n) = \frac{n}{p+n} \text{ for parent node}$$

- The question will split the examples
  - into two subsets k=1,2 as *v*=2
  - with p_k positive examples and n_k negative examples each.
- How many positive and negatives would we expect if there was no change in distribution from the training data?

$$\hat{p}_k = \underbrace{(p_k + n_k)}_{\substack{items \\ in\ split}} \underbrace{\frac{p}{p+n}}_{\substack{expected \\ +rate}} \qquad \hat{n}_k = (p_k + n_k)\frac{n}{p+n}$$

χ^2 Test

- We can look at how much our categories differ from what would be expected if the proportion of categories did not change

χ^2 test statistic is

$$\Delta = \sum_{k=1}^{\nu} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

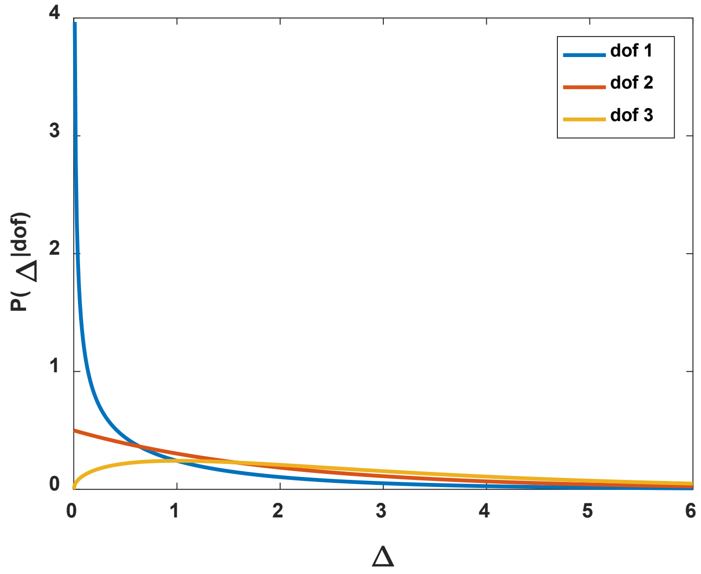
measure of deviation

where ν is the number of splits

- When Δ is small, we are close to the original distribution.

χ^2 Test

- The test statistic has a distribution that is related to the number of categories – 1. This is referred to as the *degrees of freedom* (dof) and for a binary classifier, the dof is 2-1=1.



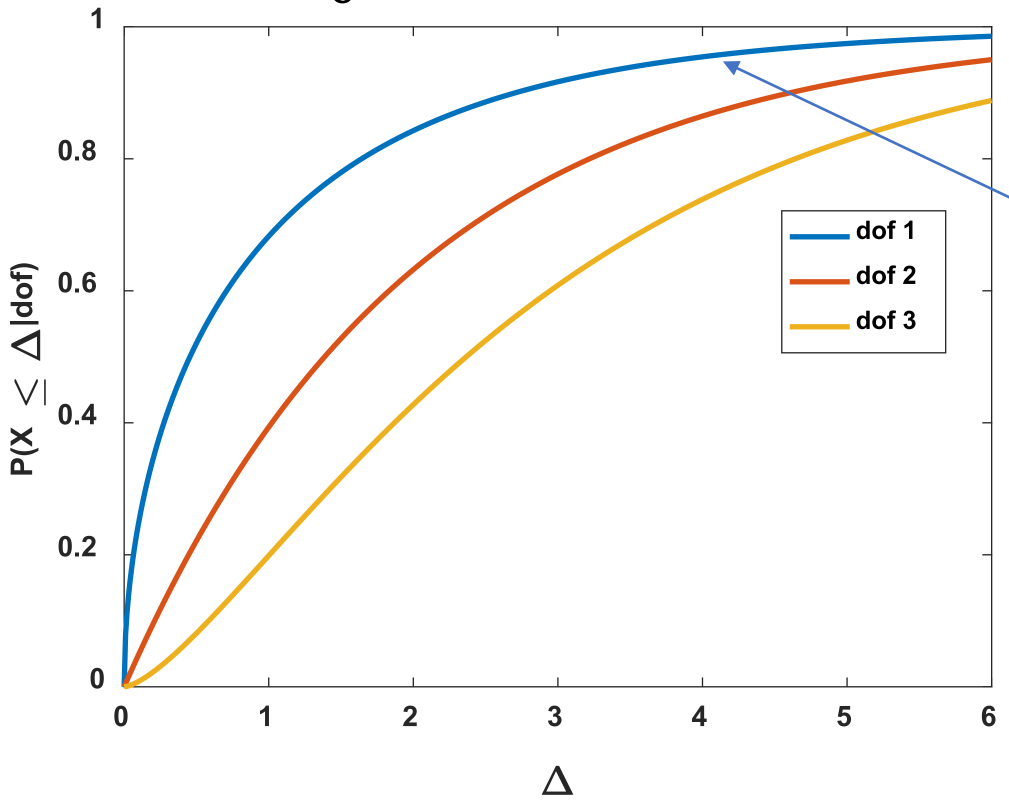
χ^2 probability density function available in scipy: scipy.stats.chi2

The formula for this is beyond our scope, but the plot shows the likelihood of having a value of Δ assuming that the distributions are identical.

Cumulative density function (CDF)

χ^2 cdf: scipy.stats.chi2.chi2cdf

- Suppose we integrate $\int_0^{\Delta} P(X|dof)$



With 1 dof,
95% of Δs
expected to
be less than 3.84

Not very likely that our split with Δ=3.84 has the same distribution as the parent

The formula for χ^2 is beyond our scope, but the plot shows the probability of having a value of Δ assuming that the distributions are identical.

χ^2 test example

Let’s return to the bird/mammal example:

	p	n	q (+ rate)	B(q)
before question	6	5	6/11	0.99
¬flies	4	1	4/5	0.72
flies	2	4	1/3	0.92

Expected in in each split if the distribution does not change: $\hat{p}_k = p \times \frac{p_k + n_k}{p+n}, \hat{n}_k = n \times \frac{p_k + n_k}{p+n}$

$$\hat{p}_{flies} = 6\frac{2+4}{6+5} = \frac{36}{11}, \hat{n}_{flies} = 5\frac{2+4}{11} = \frac{30}{11}$$

$$\hat{p}_{\neg flies} = 6\frac{4+1}{11} = \frac{30}{11}, \hat{n}_{\neg flies} = 5\frac{4+1}{11} = \frac{25}{11}$$

χ^2 test example

$\hat{p}_{flies} = \frac{36}{11}, \hat{n}_{flies} = \frac{30}{11}$

$\hat{p}_{\neg flies} = \frac{30}{11}, \hat{n}_{\neg flies} = \frac{25}{11}$

from table:

$p_{flies} = 2, n_{flies} = 4$

$p_{\neg flies} = 4, n_{\neg flies} = 1$

- Compute χ^2 statistic Δ

$$\Delta = \sum_{k=1}^{2} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

$$= \underbrace{\frac{\left(2 - \frac{36}{11}\right)^2}{\frac{36}{11}} + \frac{\left(4 - \frac{30}{11}\right)^2}{\frac{30}{11}}}_{flies} + \underbrace{\frac{\left(4 - \frac{30}{11}\right)^2}{\frac{30}{11}} + \frac{\left(1 - \frac{25}{11}\right)^2}{\frac{25}{11}}}_{\neg flies}$$

$$\approx 0.4949 + 0.5939 + 0.5939 + 0.7127$$

$$\Delta \approx 2.3956$$

χ^2 test example

We have one dof and Δ=2.3956. Significant change in distributions?

- The χ^2 cdf of a value Δ with the appropriate degrees of freedom will tell us the probability that the children do not have significant changes:

$$cdf_{\chi^2}(\Delta, 1\, dof) = 0.8783$$

- Implies
  - ~ 88% of all cases where the distribution does not change significantly will have Δ<2.3956
  - ~ 12% of distributions without a significant change have Δ≥2.3956.

- Split is probably significant, but 12% chance we are wrong.

χ^2 test

- Define an acceptable level of error called a p-value.
- Very common to use p=0.05
  (5% chance hypothesis is wrong)
- Look up threshold from χ^2 inverse cdf at $1 - p$-value:

$$\Delta_\tau = cdf_{\chi^2}^{-1}(1 - p_{value}, dof)$$

- Compute Δ for each leaf:

$$prune(\Delta, \Delta_{pval}) = \begin{cases} \Delta < \Delta_\tau & \text{prune: likely no signficant difference} \\ \Delta \geq \Delta_\tau & \text{retain: likely significant difference} \end{cases}$$

χ^2 Test

- Python does not have χ^2 routines, but the Scientific Python library does.

```
import sicpy.stats.chi2
dof = 1
p_value = .05
p05 = scipy.stats.chi2.ppf(1-p_value, dof)  # inverse CDF:  3.84
```

- Caveat: Only try this on leaf nodes of a constructed tree!
  - Sometimes, multiple levels have more power than a single one.
  - Pruning as we go can prevent us from ever seeing this.

More thoughts on decision trees

- Continuous/integer-valued attributes
    - Don’t create infinite branches
    - Select a split point
        - Sort values
        - Keep running total of number of +/- examples for each point in sorted list and pick the separating point that gives the best separation.
- See text for information on multivalued attributes and continuous-valued outputs.

Decision tree summary

Relatively straight-forward learners that

- recursively partition the feature space into hyperplanes,
- are sensitive to overtraining, but have methods to prune,
- and are **easy for humans to understand**

Image credit: Sara France, UCI Med School

Do I have a good hypothesis function?

- Assume data are *independent* and *identically distributed* (**iid**)
  - Independent – Examples $e_j=(x_j,y_j)$, $e_k=(x_k,y_k)$ are unrelated to one another when $j \neq k$.

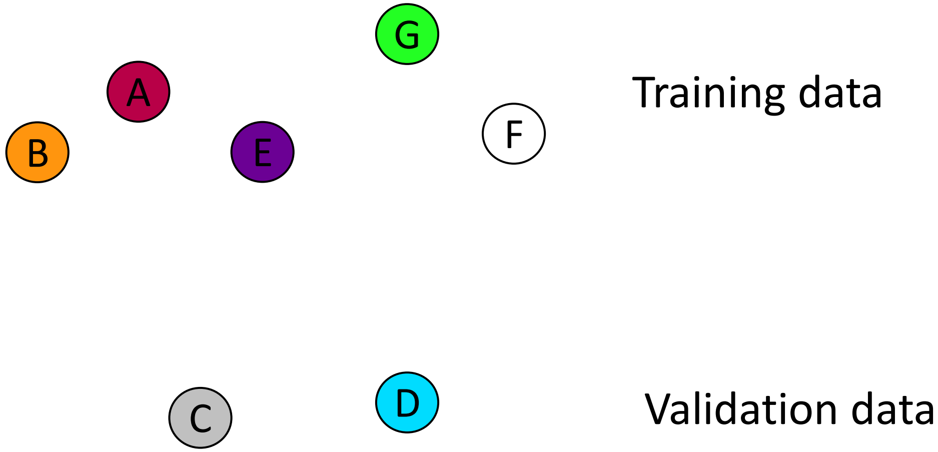
$$P(E_j \mid E_k) = P(E_j)$$

  - Identically distributed – Whatever process generated e_j is also responsible for generating e_k and did not change.

Warning: iid assumptions do not always hold!

Do I have a good hypothesis function?

- We cross-validate the learner on a separate validation set

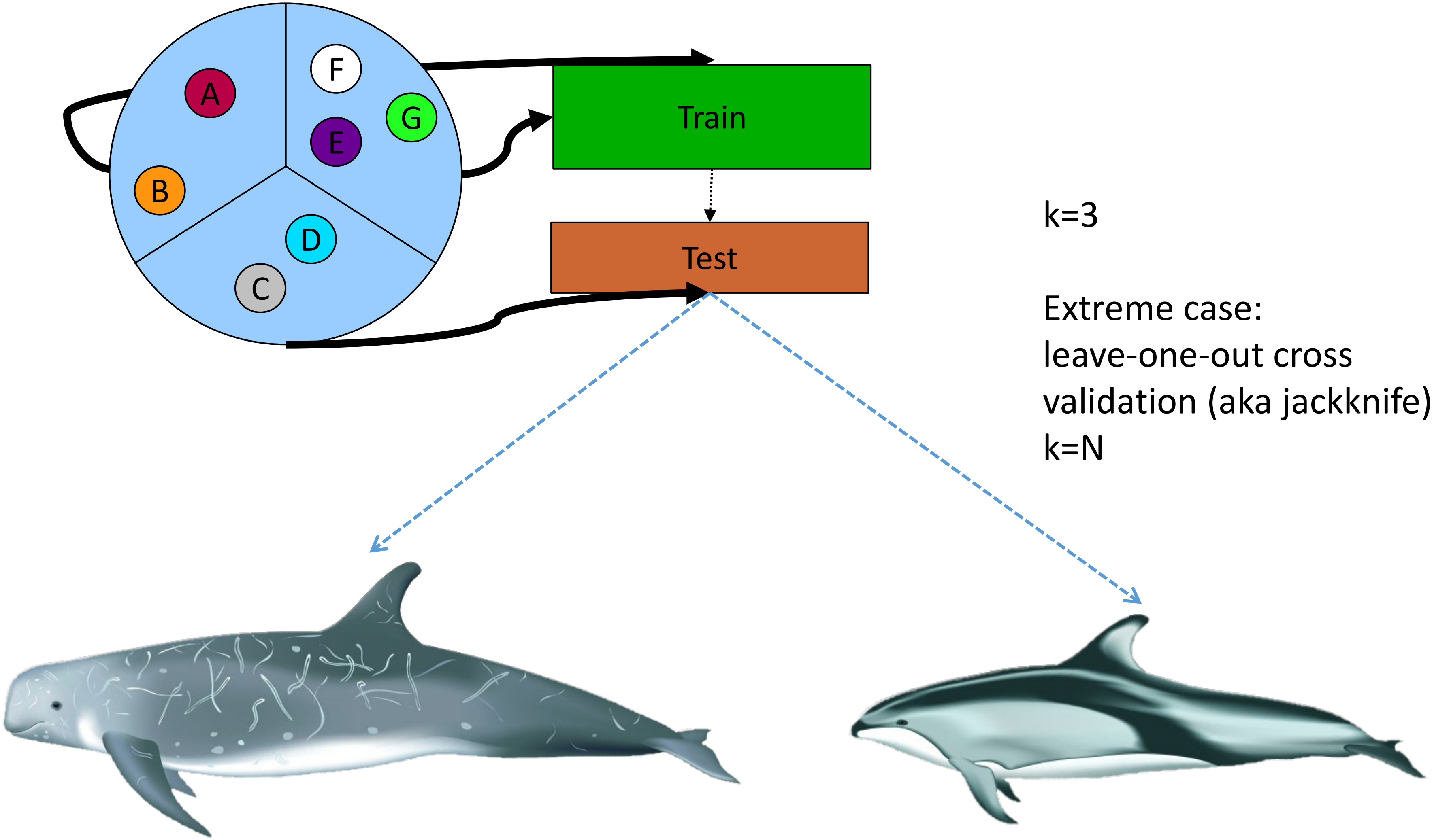


Training data

Validation data

- Problem: We don’t exploit all our data

k-fold cross validation



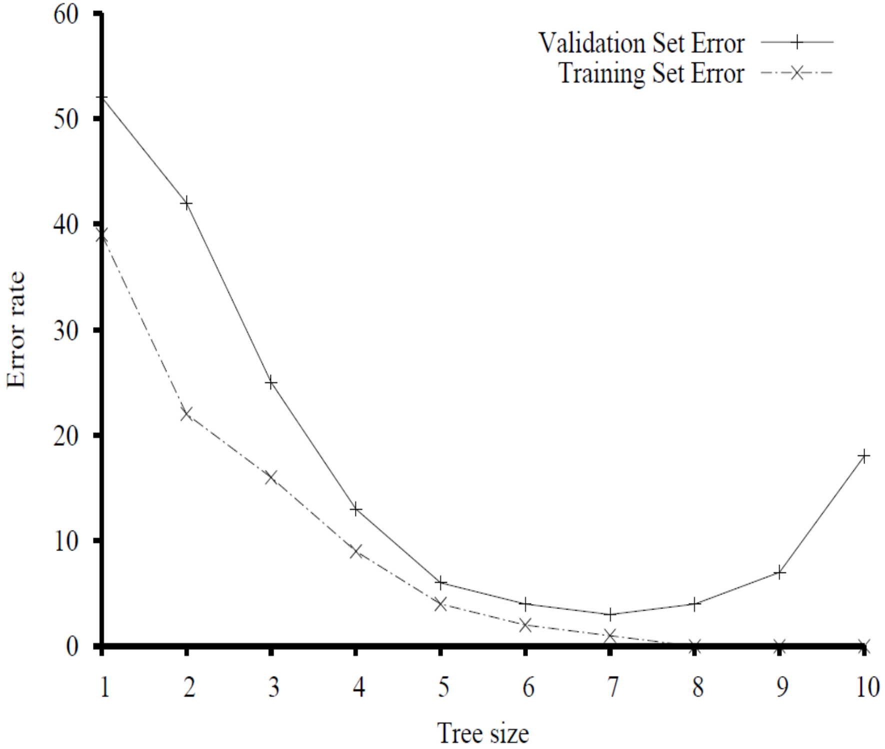
k=3

Extreme case:
leave-one-out cross validation (aka jackknife)
k=N

slide courtesy Simon Qiu

Model selection

- More complex models (e.g. more nodes in a decision tree) learn the training data better, but are they really better?
- For this, we look at validation error



Note: There are also statistics that can help us select models (beyond our scope)

Loss and the North Pacific Right Whale

image: NOAA

Does optimizing misclassification rate make sense?

Loss

- Loss functions are a form of utility function that provide a cost for misclassification

$$L(x, y, \hat{y}) = \text{cost(predicting h(x)=}\hat{y}\text{ given f(x)=}y\text{)}$$

- Suppose that it so useful to find a right whale that we do not mind misclassifying a bunch of non right whales as whales

$$L(x, y = \text{right whale}, \hat{y} = \text{other}) = 10$$
$$L(x, y = \text{right whale}, \ \hat{y} = \text{right whale}) = 0$$
$$L(x, y = \text{other}, \hat{y} = \text{right whale}) = 1$$
$$L(x, y = \text{other}, \hat{y} = \text{other}) = 0$$

Loss

- Some learners attempt to minimize loss

- Common loss functions

$$L_1(x, y, \hat{y}) = \left| y - \hat{y} \right| \qquad \text{absolute loss function}$$

$$L_2(x, y, \hat{y}) = \left(y - \hat{y} \right)^2 \qquad \text{squared loss function}$$

$$L_{0/1}(x, y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & \text{otherwise} \end{cases} \qquad \text{0/1 loss function}$$

Generalization loss

- What is our loss when we use a novel data set ε?
- The expected loss requires the distribution of (X,Y) which we probably do not have:

$$GenLoss_L(h) = \sum_{(x,y)\in \grave{o}} L(x,y,h(x))P(x,y)$$

but we can estimate it empirically on a finite set of examples *E* of *N* samples:

$$EmpLoss_L(h) = \frac{1}{N}\sum_{(x,y)\in E} L(x,y,h(x))$$

Note: Generalization loss is frequently referred to as *risk*

Generalization loss

- Selection of our learner *h** now becomes:

$$h^* = \underset{h \in H}{\arg\min}\, EmpLoss_{L,E}(h)$$

- Are we guaranteed *h** = *f*? No:
  - Unrealizability: *f* may not be in *H*
  - Variance: Learners return different f’s for different training sets
  - Noise:
    - f may be noisy (e.g. stochastic component – different y’s for the same x)
    - The training samples may have mis-measured attributes or incorrect labels
    - Might not have measured important attributes.
  - Complexity: Learner may not achieve a global minimum.

Regularization

- Occam’s Razor states less complex models are better.
- Can we incorporate this into our model selection?

$$\text{Cost}(h) = EmpLoss(h) + \lambda Complexity(h)$$

$$h^* = \underset{h \in H}{\arg\min}\, \text{Cost}(h)$$

- The cost function is called a regularization function

Complexity models are beyond our scope, but if you want to know
more read about MDL in chapter 20 or information criteria (e.g. AIC, BIC)

Reducing model complexity

- Learner complexity can be reduced by pruning the feature space:
  - feature selection
  - principle components analysis
  - nonlinear dimension reduction

You are not responsible for 19.5 – The Theory of Learning

Non-parametric models

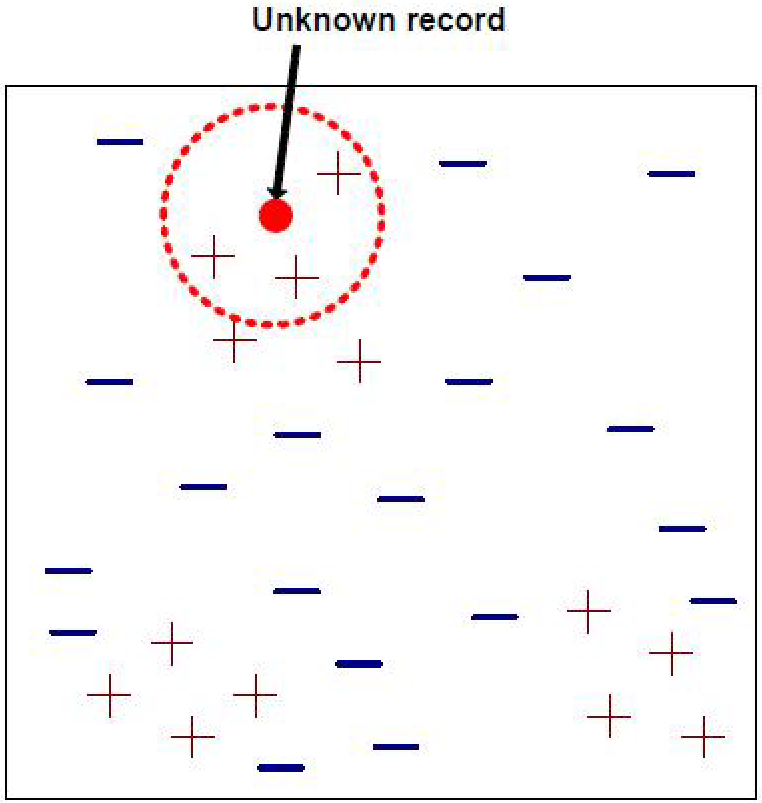
- Neural nets and decision trees have models with parameters
  - decision node parameters: attribute and cut-point/categories for sub-trees
  - neural nets: weights and connections
- Non-parametric models
  - Cannot be characterized by a *bounded* set of parameters
  - Simplest case:
    Look at every example and use it to classify a novel example.
    (Parameters $\propto$ #training examples)
  - Called instance- or memory-based learning

Nearest neighbor models

- Use a distance metric to find the k closest neighbors, e.g. for continuous attributes:

$$L^P(\vec{x}_j, \vec{x}_q) = \left(\sum_{i=1}^{D} |x_{j,i} - x_{q,i}|^p \right)^{\frac{1}{p}}$$

- Use the plurality (majority) of labels that are the k closest



credit: humanoriented.com

Nearest neighbor models

- The good
  - Simplicity
  - Effective technique for low-dimensional data
- The Bad – Searching is expensive with large training sets, but we can mitigate for this:
  - trees – Similar to a decision tree (split on value, may at times need to search both sides)
  - Locally sensitive hash tables
    - Hash functions
      - set of projections on to lines (similar to linear classification examples)
      - Line projections are discretized into buckets
    - Can be much more effective than tree approach

Nearest neighbor models

- and the Ugly
  - N points uniformly distributed in an $\Re^D$ unit hypercube.
  - To capture r=.01 of the observations, what edge length *l* would we need in a random sample?

  - Samples are randomly distributed and total volume is 1, so we need a volume of r (.01).

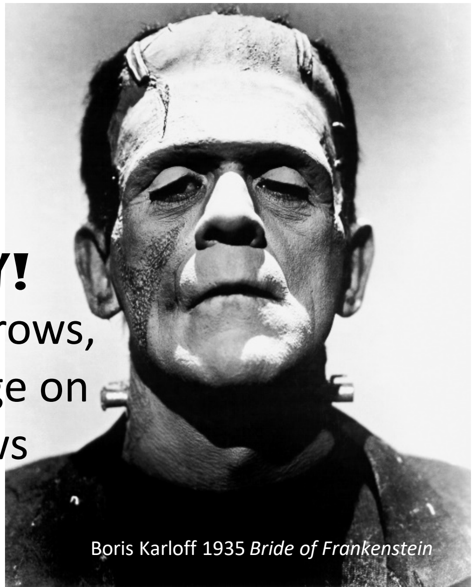
  - $d = 1 \rightarrow l = .01^{\frac{1}{D}} = .01$

  - $d = 10 \rightarrow l = .01^{\frac{1}{10}} = .63$

  - $d = 100 \rightarrow l = .01^{\frac{1}{100}} = .96$

$$l^d = r \rightarrow l = r^{1/D}$$

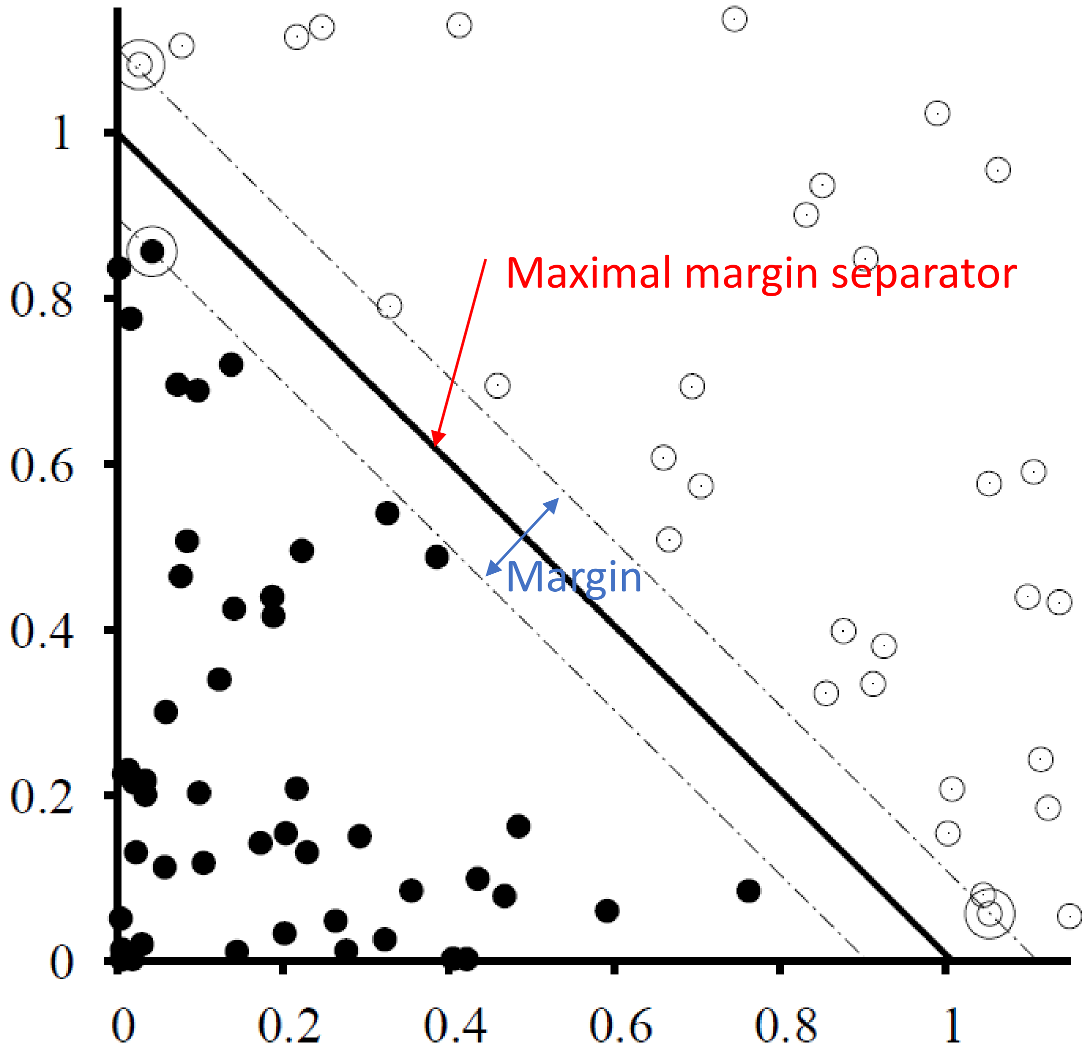
THE CURSE OF DIMENSIONALITY!
As the dimension grows, the size of each edge on the hypercube grows as well!



Boris Karloff 1935 *Bride of Frankenstein*

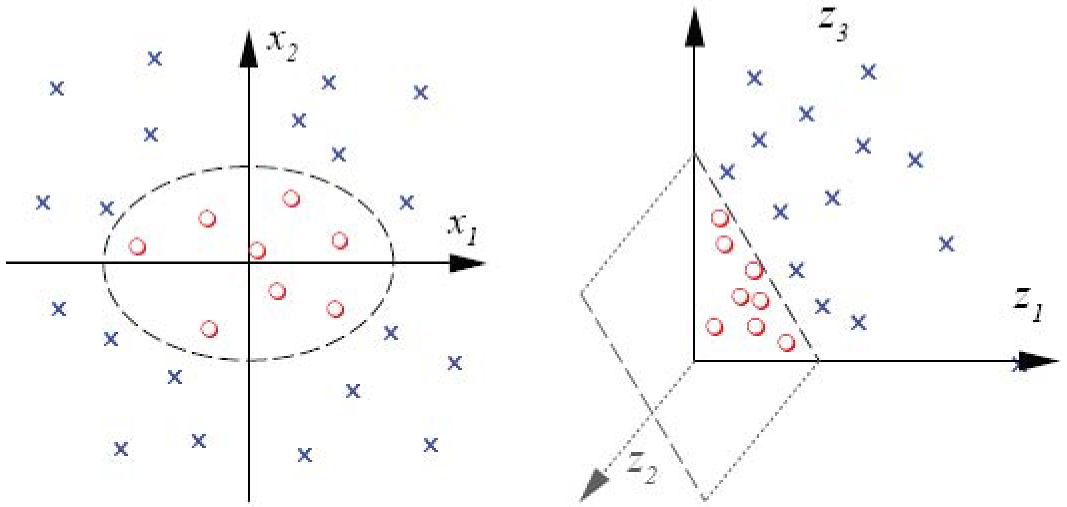
Support vector machines (SVMs)

- A margin is the distance to the closest examples on either side of a hyperplane.
- SVM approaches attempt to maximize the margin



Support vector machines

- Can only separate linear problems, but a kernel function can project the data into a higher dimensional space where perhaps the data can be better separated



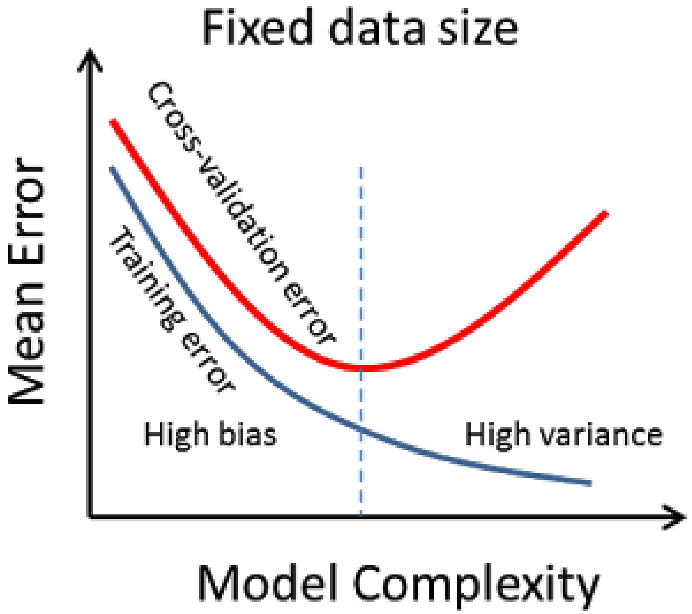
credit: Chris Thornton, Univ. Sussex

Support vector machines

- Maximal margins com are computed as functions of training examples
- Consequently
  - SVMs are nonparametric techniques
  - In practice, only a small subset of training examples, the support vectors, are required
- The training algorithm is beyond our scope, but is essentially an optimization problem.

Bias and variance

- Error in learning comes from two sources: bias and variance



Ricky Ho, horicky.blogspot.com

Bias – Large when learners make consistently incorrect predictions
Variance – Large when different training sets result in different predictions

Ensemble learning

- Ensemble learners frequently are a collection of weak learners that are combined to form a robust classifier

  Weak learner – A simple learning algorithm that is likely to have a high bias (e.g. a single node, or stump, of a decision tree)

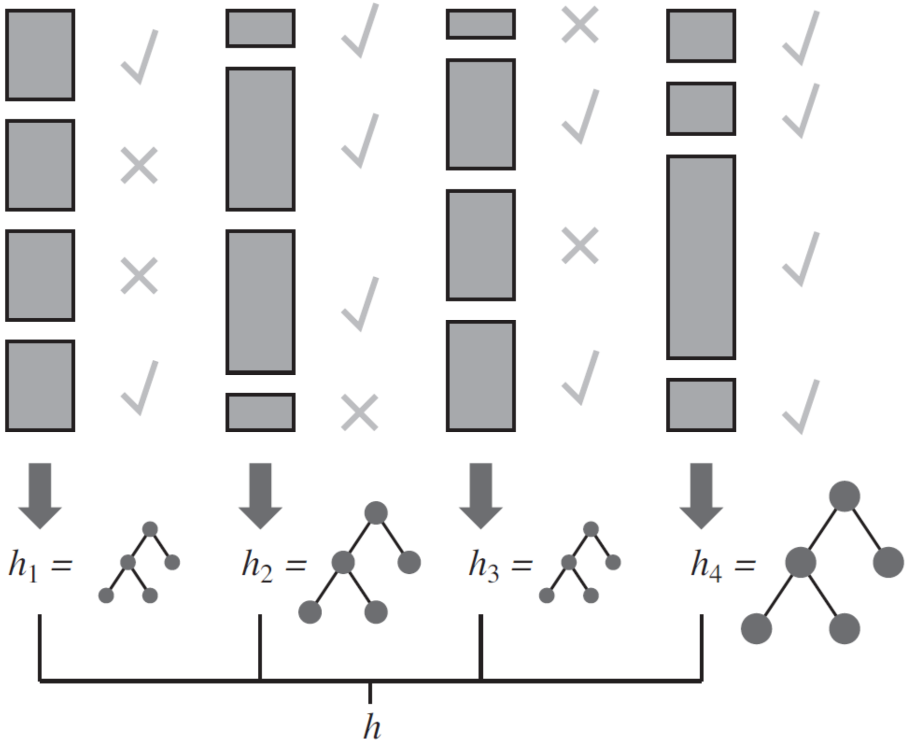
- Ensemble learners typically use collections of weak classifiers to reduce both bias and variance.

Adaptive Boosting (ADABOOST)

- Type of ensemble learning algorithm
- Use decision stumps as the weak learner
- Examples are weighted. Loss is greater for examples with higher weight

Adaptive boosting

- Start with uniform weights
- Learn the decision tree stump
  - Redistribute weights: misclassified training examples get more weight
  - Produce a classification weight as a function of error
  - Iterate until k learners are produced



Adaptive boosting

- Classification
  - Classify an example by each of the k weak learners
  - Use plurality of *weighted* decisions

- A very interesting tidbit…
  Letting k grow even after the ADABOOST fits training data perfectly frequently results in slightly improved generalization scores.

  Some interpret this as ADABOOST being robust to overtraining.

Unsupervised methods

(not in book)

- Key idea: group things that are similar together

- What gets grouped depends on a similarity/dissimilarity measure, e.g.:

$$d(x, y) = (x - y)^2$$

- What do you think?
  - Group speech by pitch?

Similarity

- How similar are two vectors?

- Distance metric (distortion)
  - $d(x, y) = \begin{cases} 0 & x = y \\ > 0 & x \neq y \end{cases}$

  - Satisfies triangle $\neq$: $d(x, y) + d(y, z) \geq d(x, z)$

  - Symmetric: $d(x, y) = d(y, x)$

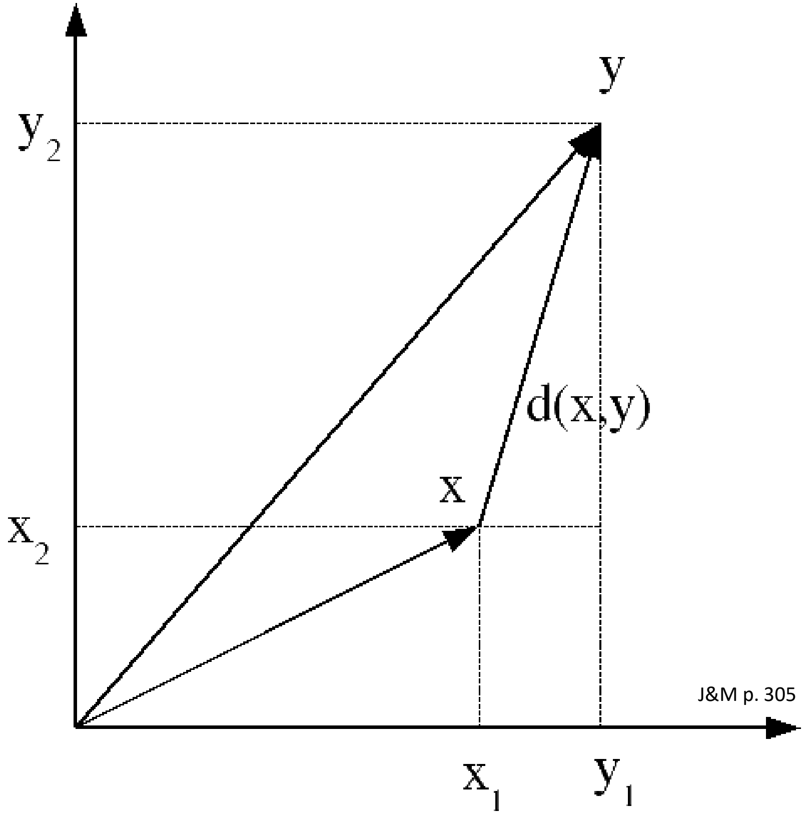
Euclidean distance/distortion

Straight line distance (squared) between two points

$$d^2(x,y) = \sum_{i=1}^{D} (x_i - y_i)^2$$

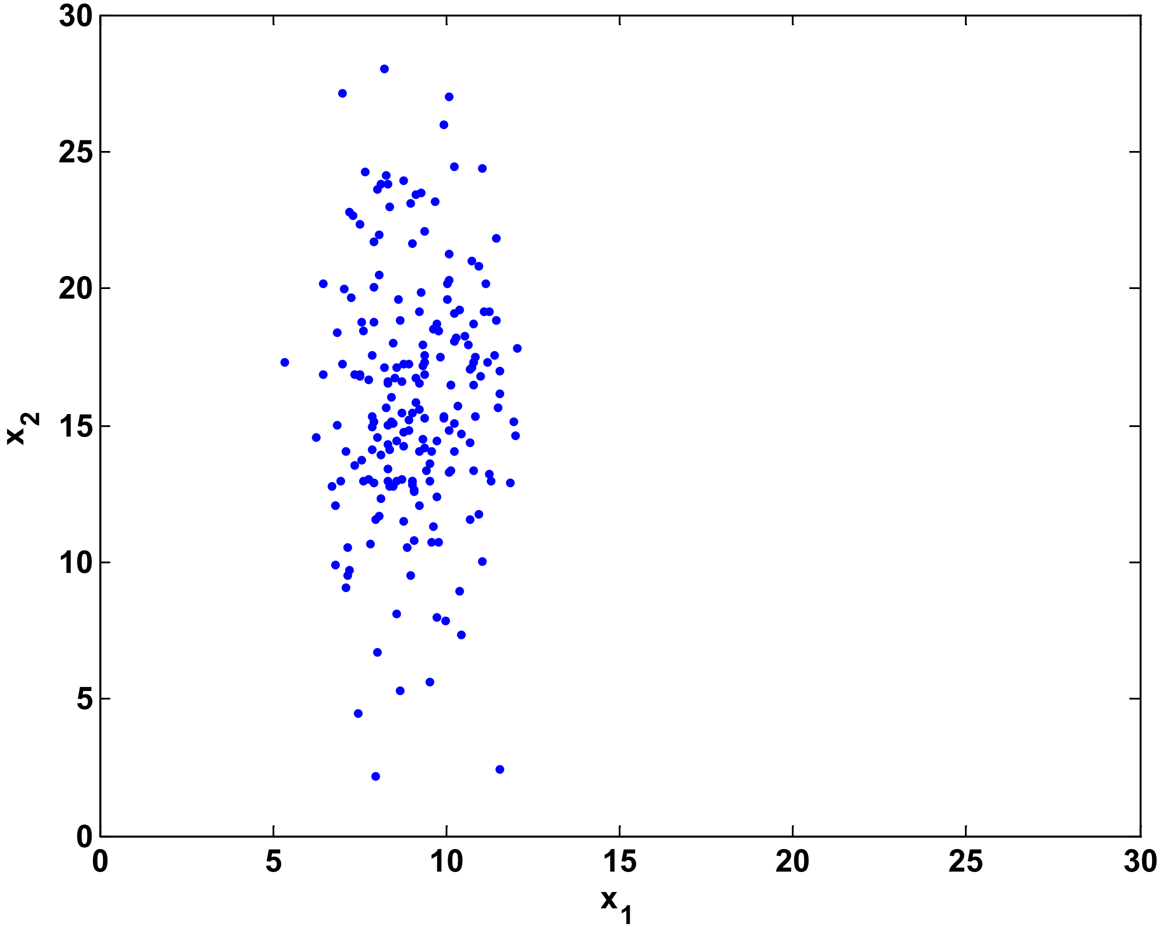
as a vector operation:

$$d^2(x,y) = (\text{x-y})^T (x - y)$$



J&M p. 305

Does Euclidean distance always make sense?

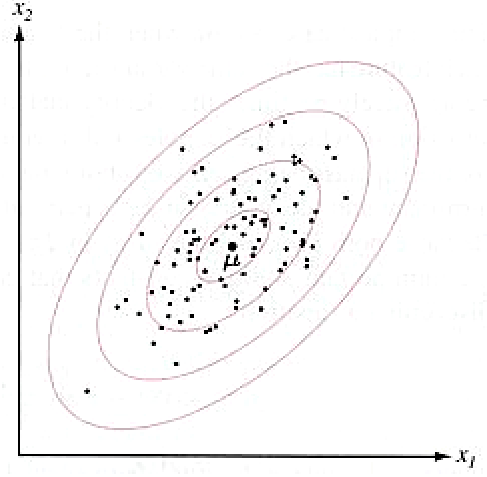


Scaling variables

- When different features do no have the same range or variance, they can be difficult to compare
- Common technique is to z-normalize a *z-score*.
  - $z = \frac{x - \mu}{\sigma}$
  - z normalization
    - For normally distributed data (bell curve), transforms to a normal distribution with mean 0 and variance 1.
    - $n(\mu, \sigma^2) \rightarrow n(0,1)$
- This works well if features are independent of one another

Distortion - Mahalanobis

- Mahalanobis distortion
  - Accounts for the variance and covariance (Σ)
  - Removes assumption of equal scaling



$$d_{Mahalanobis}(\vec{x},\vec{y})$$
$$= (\vec{x}-\vec{y})^t \Sigma^{-1} (\vec{x}-\vec{y})$$

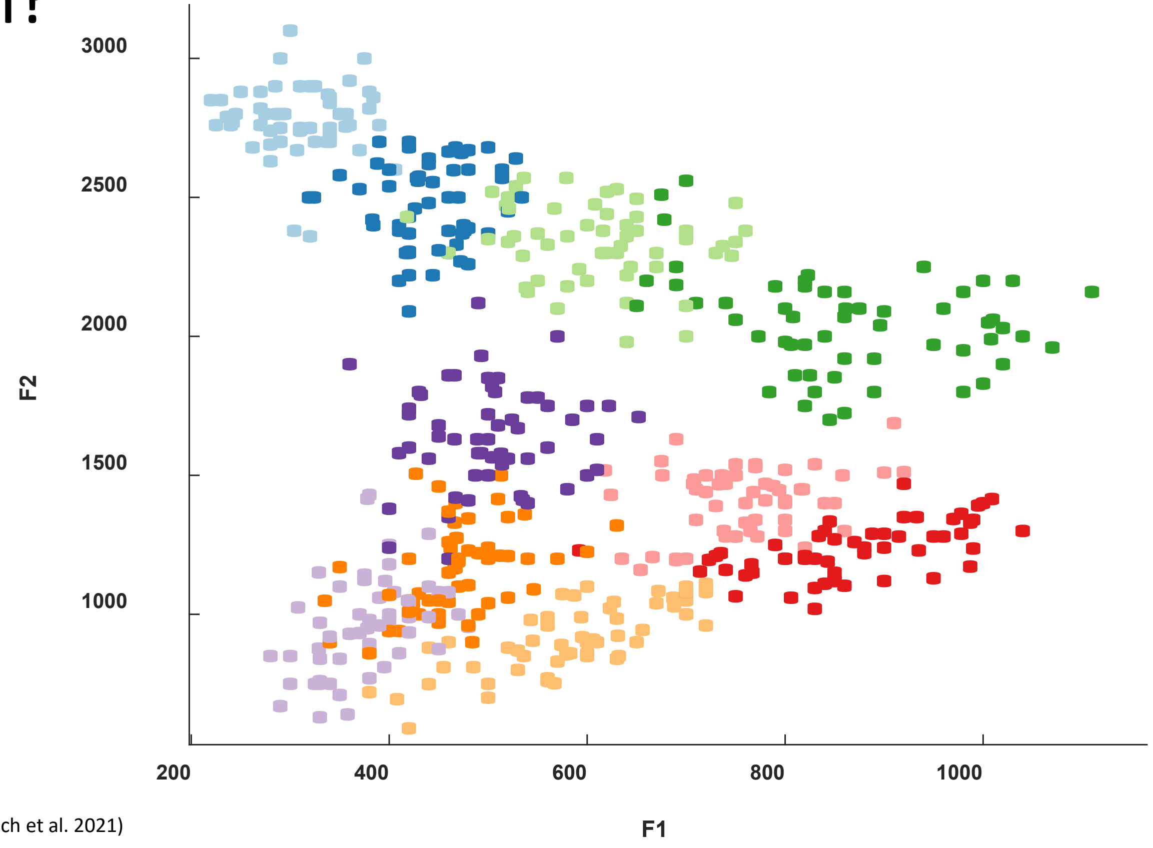
$$\Sigma = \begin{bmatrix} var(x_1) & \cdots & cov(x_d, x_1) \\ \vdots & \ddots & \vdots \\ cov(x_1, x_d) & \cdots & var(x_d) \end{bmatrix}$$

k-means not in text, See Dean Moore's explanation as a secondary source

k-means clustering

also known as vector quantization

- Let us assume that we know there are *k* clusters.
- How do we find them?

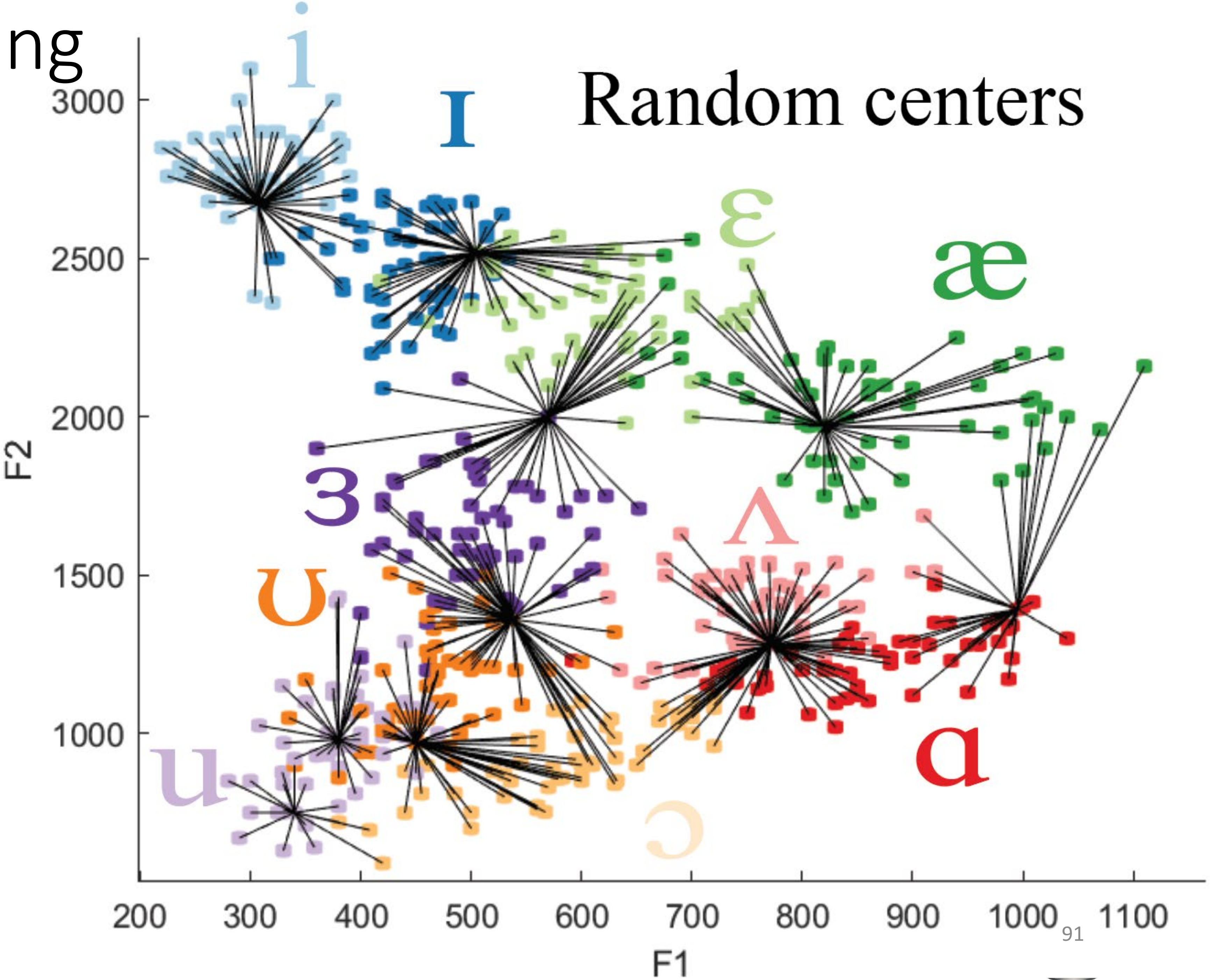


Vowel formant data

(Peterson and Barney, 1952; adapted from Roch et al. 2021)

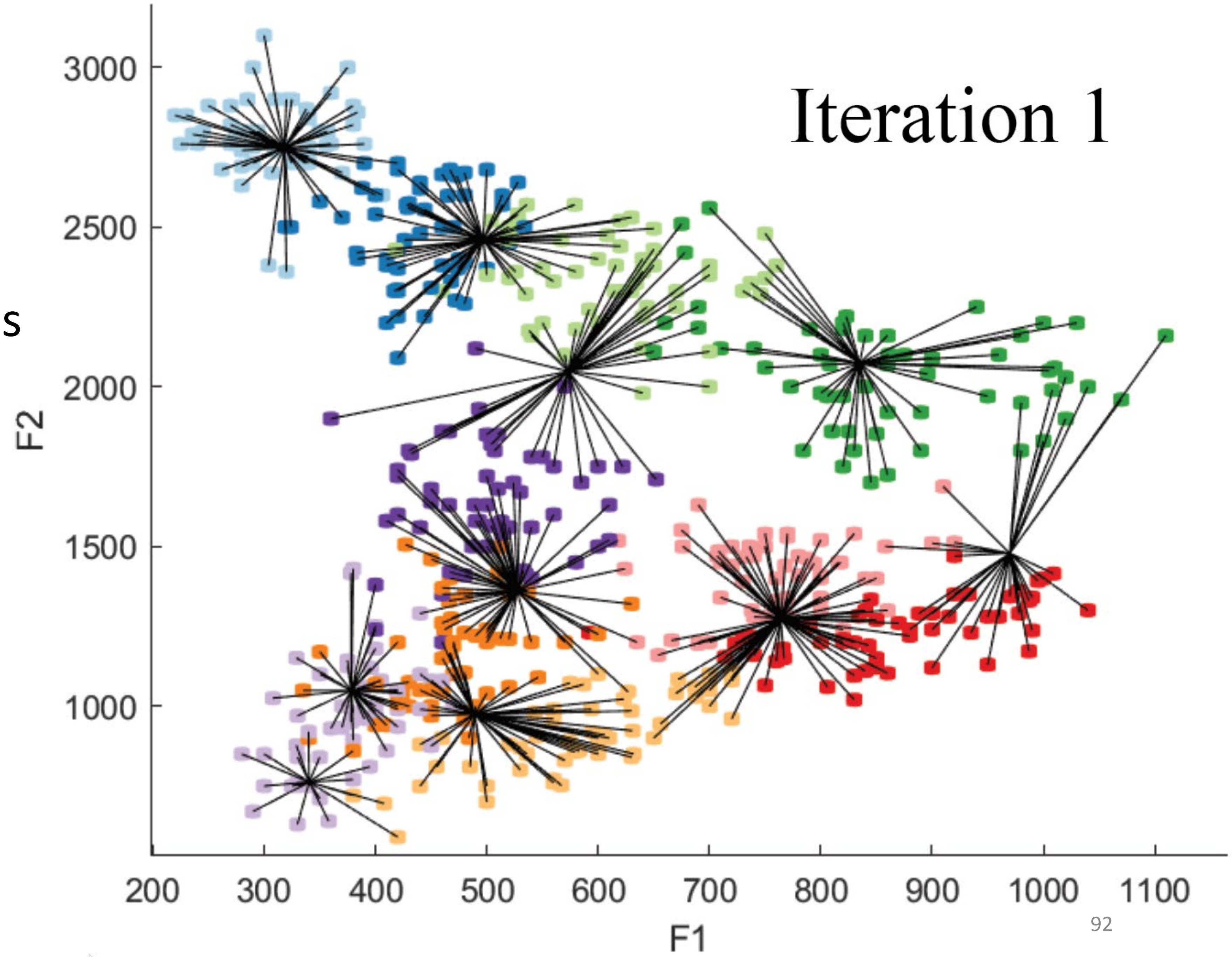
k-means clustering

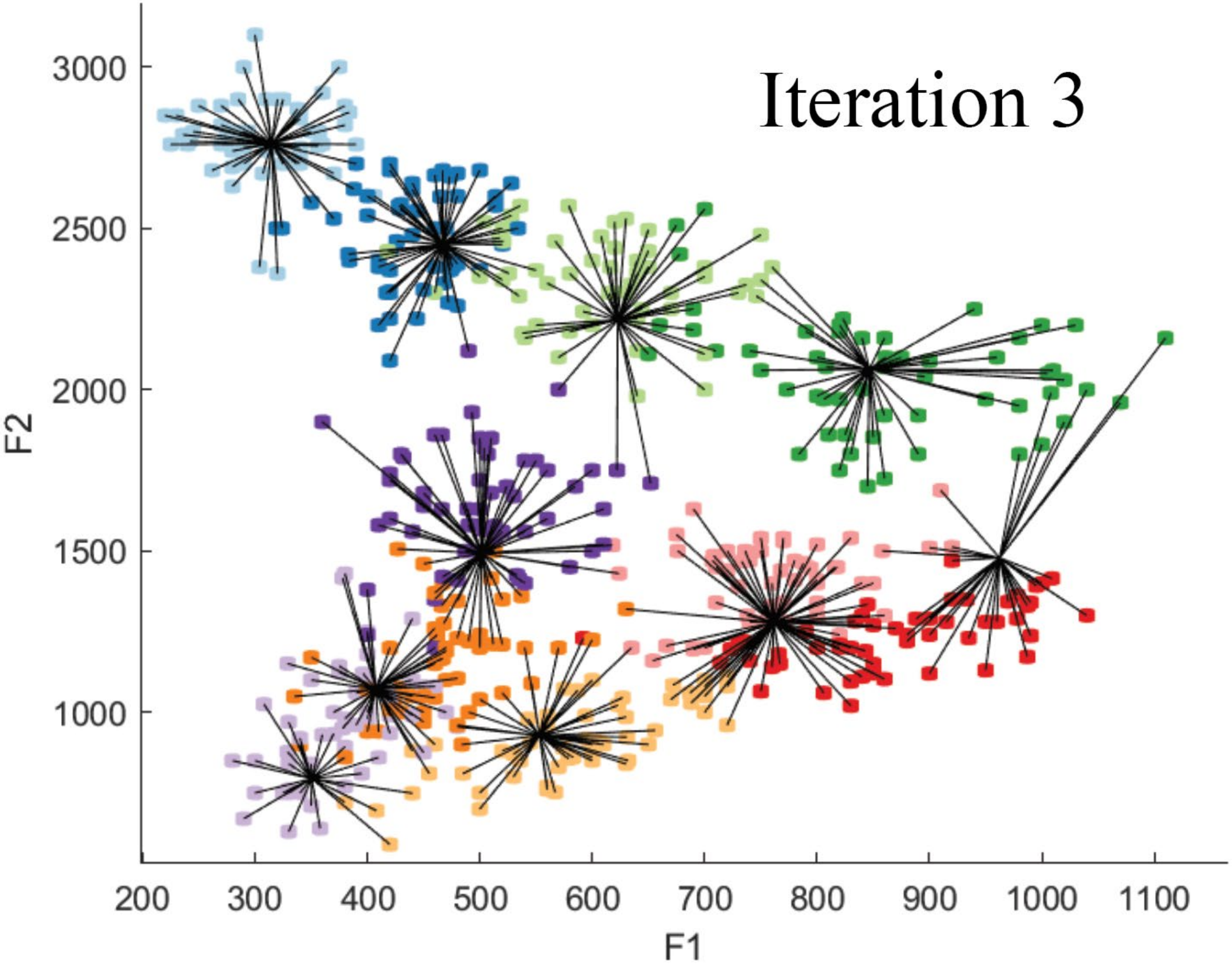
- Pick k random centers
- Find the samples closest to each one
- Closest might be measured with Mahalanobis, z-norm, Euclidean, etc.



k-means

- Compute centers and update means
- Repartition

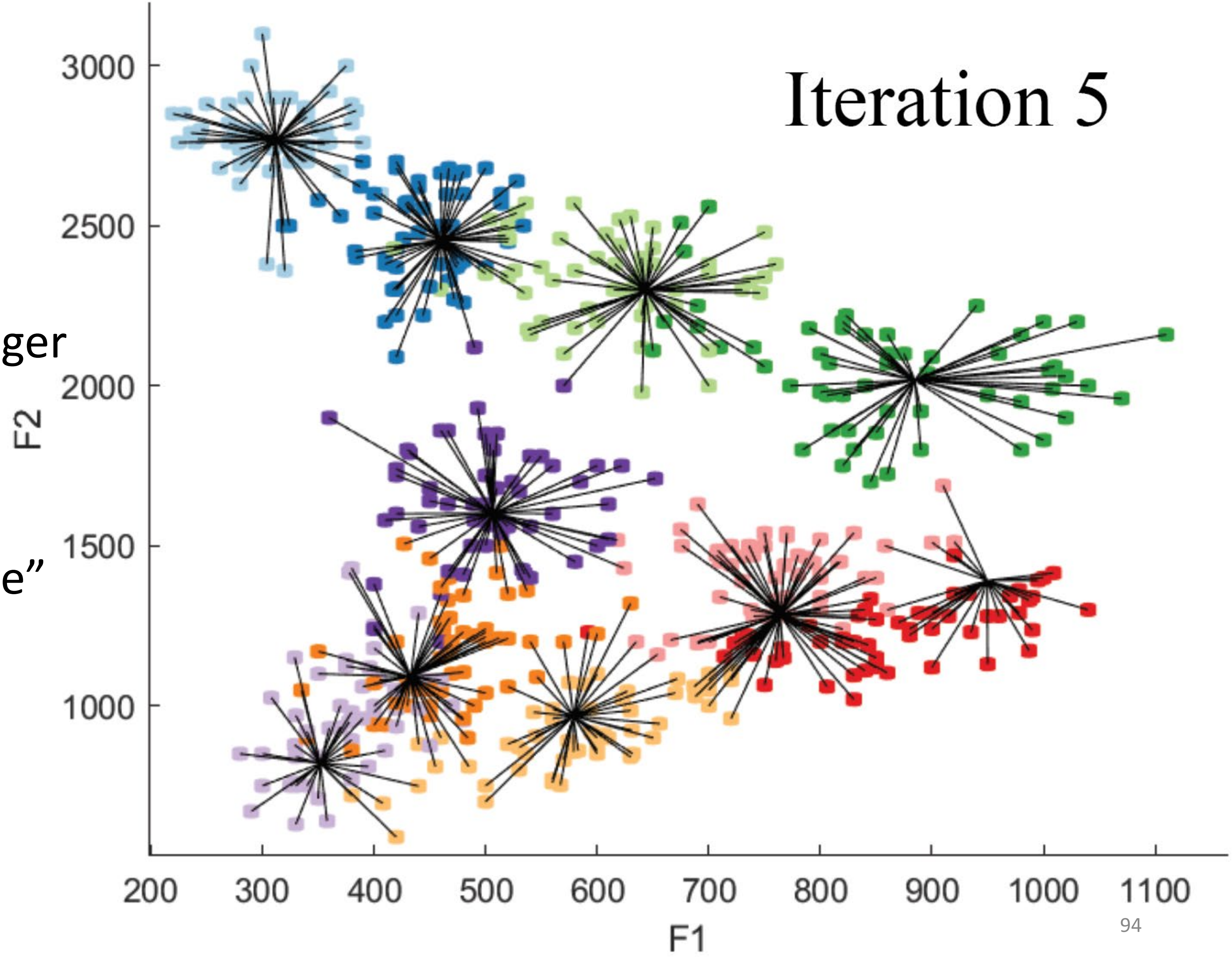



3000
2500
2000
1500
1000
F2
200
300
400
500
600
700
800
900
1000
1100
F1

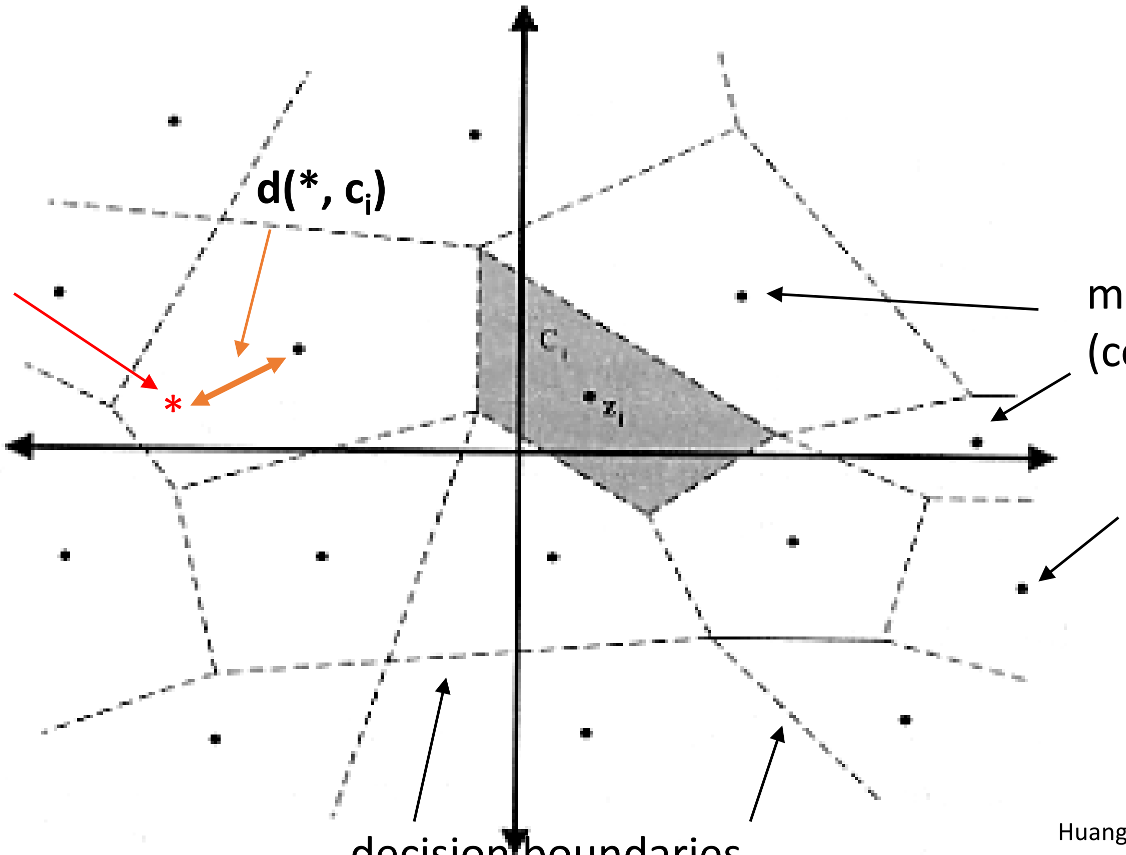
Iteration 3

k-means

- Continue until there is no longer significant changes
- Means are "representative"



R^2 partition induced by *k*-means



vector
to be
quantized

d(*, c_i)

mean vectors
(codewords)

decision boundaries

Huang et al., p 165

k-means/Vector Quantization clustering

```
Select $k$ vectors at random as initial centers from training sample X
done = false;
old_distortion = $\infty$
while not done
    Compute $d(x_i , c_j)$ for each training vector and center
    Partition training vectors according to $c_j$ which produced smallest distortion
    Compute new centers by taking the mean (centroid) of each partition
    distortion =  compute avg. minimum distortion for all training vectors
    done = distortion / old_distortion > threshold
    old_distortion = distortion
```

Quantizing

Quantization finds the closest codeword in codebook *c*:

$$q(\vec{x}, c) = \vec{c}_i \leftrightarrow i = \arg \min_{1 \le k \le K} d(\vec{x}, \vec{c}_k)$$

Sometimes we want the distortion to the closest codeword:

$$\text{distortion}(\vec{x}, c) = \min_{1 \le k \le K} d(\vec{x}, \vec{c}_k)$$

Using k-means

- Unsupervised classifier
  - Centroids represent the distribution of items
  - Each mean symbolizes a cluster
- Supervised classifiers
  - Construct multiple k-means codebooks (one per class)
  - Find class with minimum distortion

A Supervised k-means classifier

- Training

  For each class ω$_i$ in Ω construct a codebook: CB$_1$, CB$_2$, CB$_3$, …

- Testing

  Given a set of test vectors $X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_T\}$

  Find codebook with smallest distortion across all vectors

VQ Classification

```
$MinDistortion = \infty$
for $cidx$ = 1 to $|\Omega|$
  $SumDistortion = 0$
  for $vidx$ = 1 to $T$
    $SumDistortion = SumDistortion + distortion(\vec{x}_{vidx}, book_{cidx})$
  if $SumDistortion < MinDistortion$
    $MinDistortion = SumDistortion$
    $MinIdx = cidx$
Decide that $X$ belongs to class $\omega_{MinIdx}$
```

Note: Frequently, the average distortion is computed.