Unsupervised Learning

Clustering

- In *clustering*, the target feature is not given.
- Goal: Construct a natural classification that can be used to predict features of the data.
- The examples are partitioned in into *clusters* or *classes*.
- Each class predicts values of the features for the examples in the class.
- In *hard clustering*, each example is placed definitively in a class.
- In *soft clustering*, each example has a probability of belonging to each class.
- The best clustering minimizes an error measure.

EM Algorithm

- The EM (Expectation Maximization) algorithm is not an algorithm, but is an algorithm design technique.
- Start with a hypothesis space for classifying the data and a random hypothesis.
- Repeat until convergence:
  - **E Step**. Classify the examples using the current hypothesis.
  - **M Step**. Learn a new hypothesis from the examples using their current classification.
- This can get stuck in local optima; different initializations can affect the result.

k-Means Algorithm

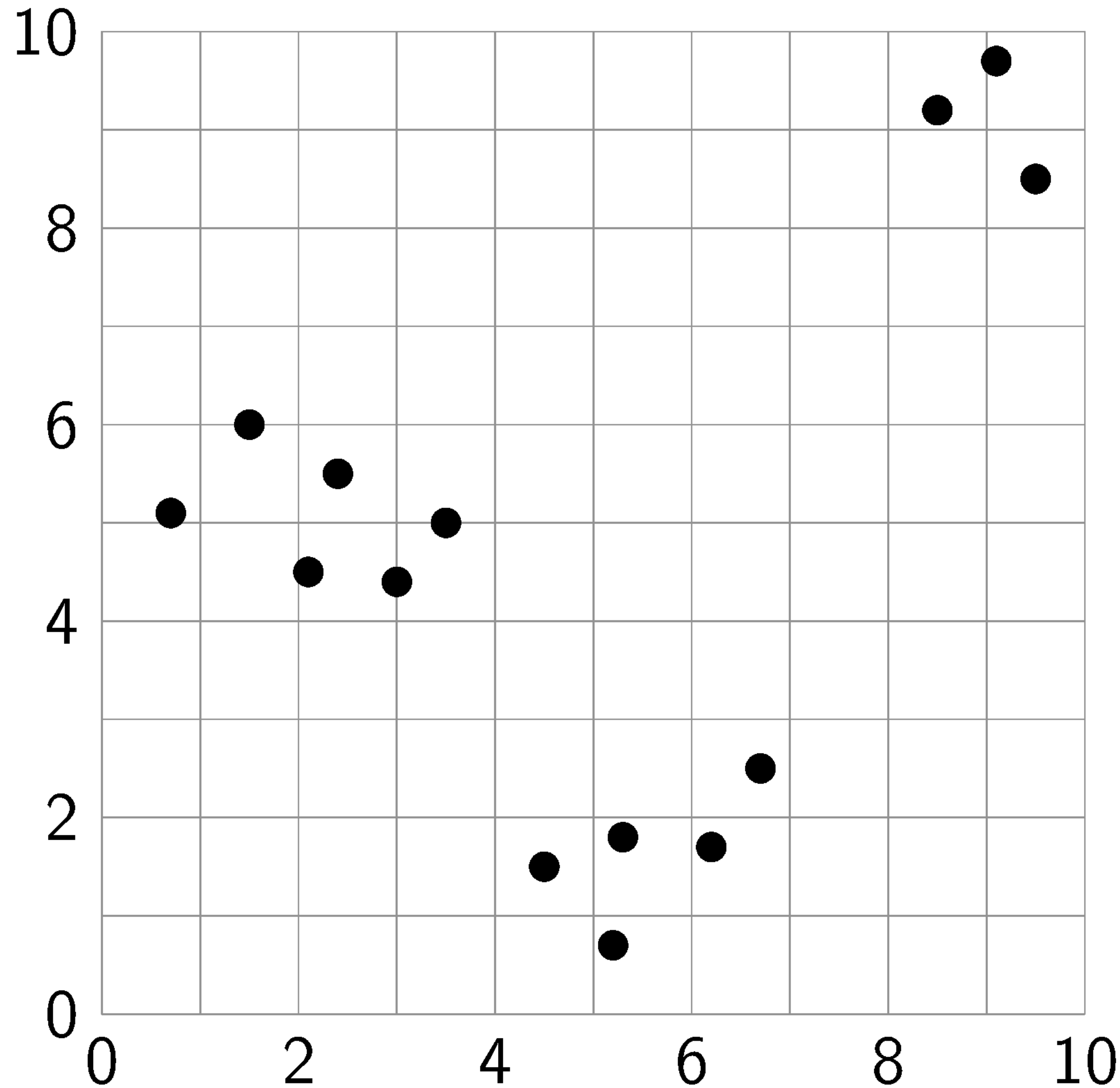
- The k-means algorithm is used for hard clustering.
- Inputs:
  - training examples
  - the number of classes/clusters, k
- Outputs:
  - Each example is assigned to one class.
  - The average/mean example of each class.
- If example $e = (x_1, \ldots, x_n)$ is assigned to class i with mean $u_i = (u_{i1}, \ldots, u_{in})$, error is

  $\|e - u_i\|^2 = \sum_{j=1}^{n} (x_j - u_{ij})^2$

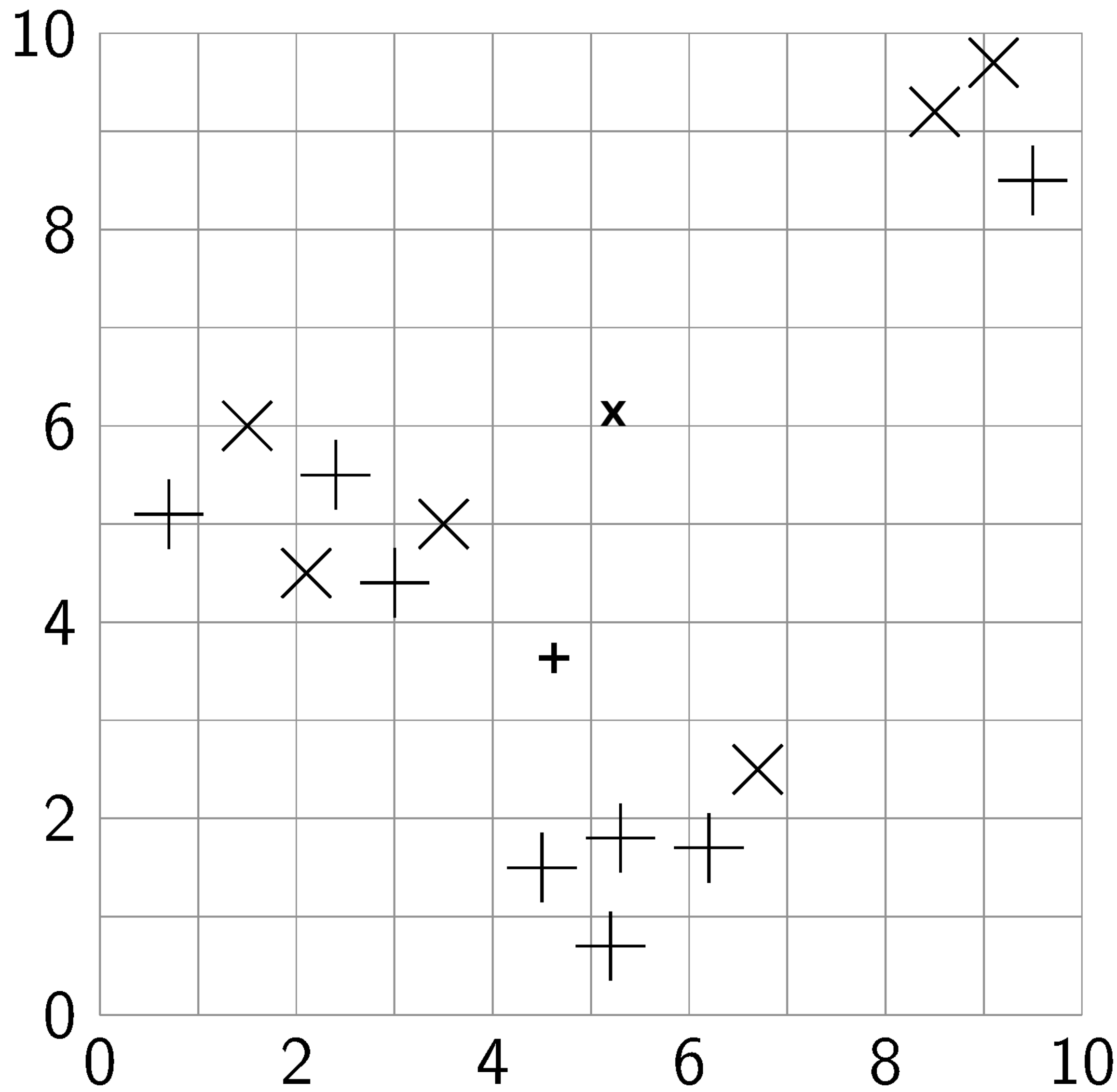
k-Means Procedure

Procedure *K-Means*(E, k)
  Inputs: set of examples and number of classes
  Randomly assign each example to a class
  Let E_i be the examples in class i
  Repeat
    M-Step:
      for each class i from 1 to k
        $u[i] \leftarrow \sum_{e \in E_i} e / |E_i|$
    E-Step:
      for each example e in E
        put e in class $\arg\min_i \|u[i] - e\|^2$
  until no changes in any E_i
  return u and E_i clusters

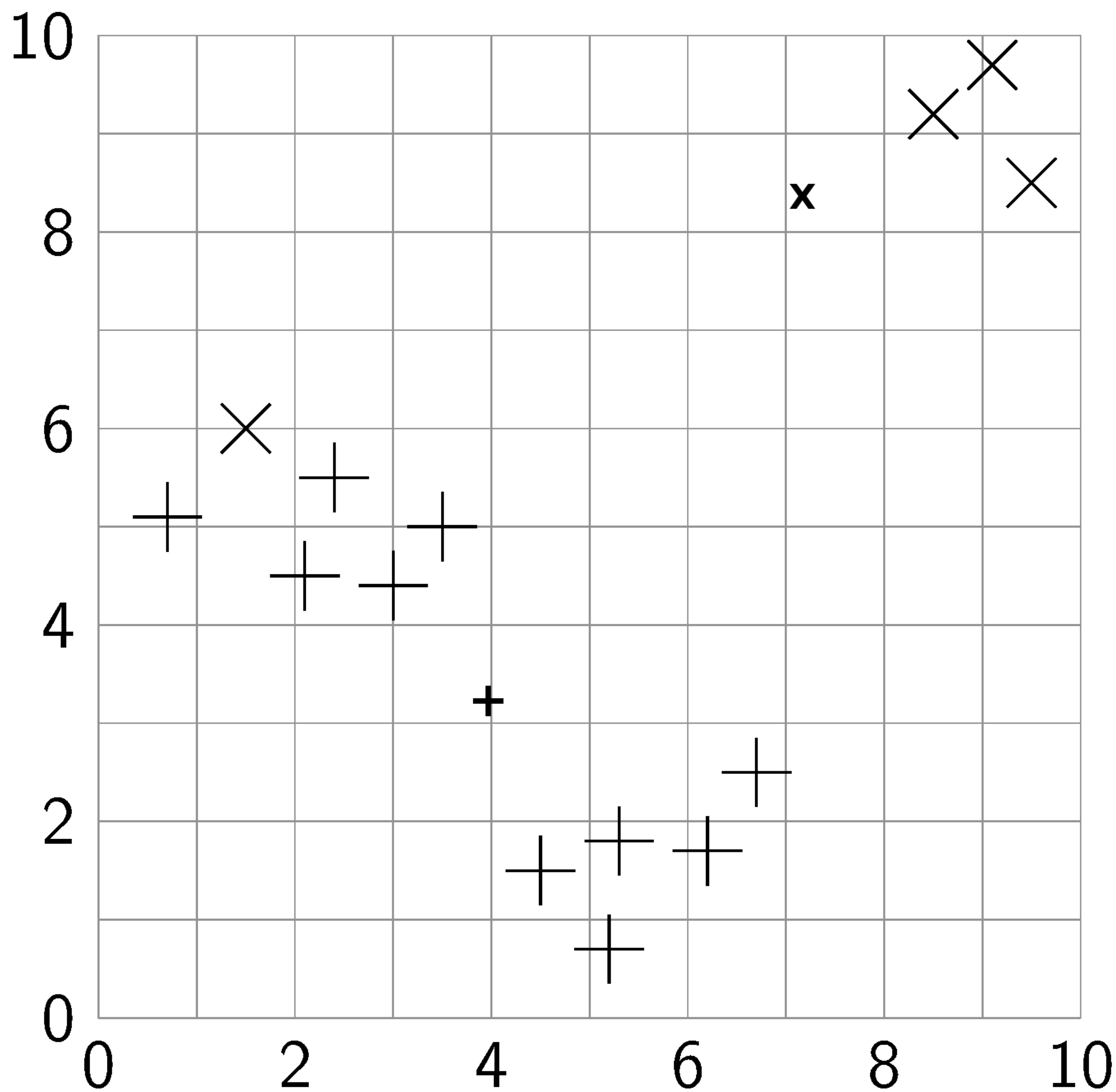
Example Data



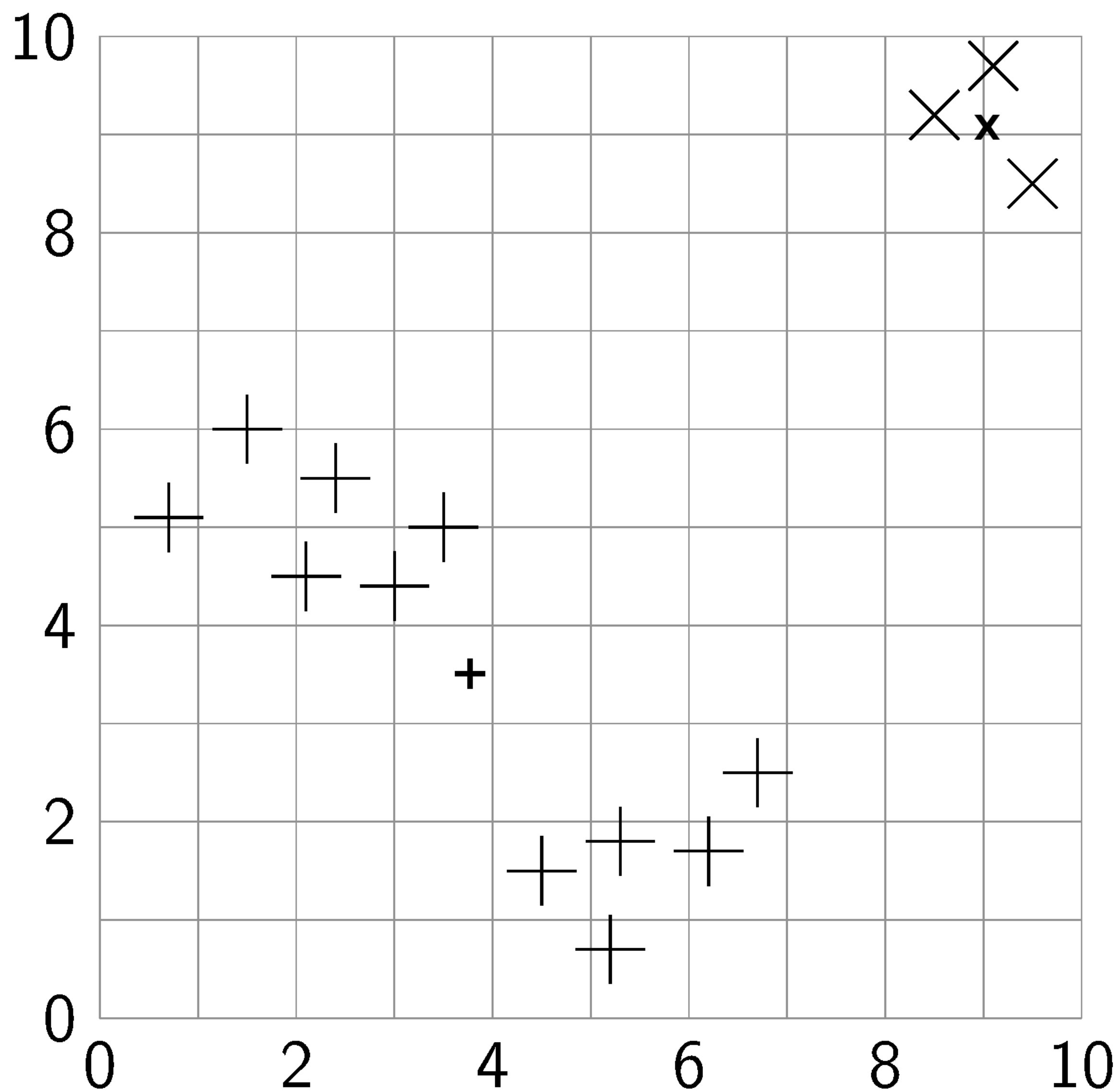
Random Assignment to Classes



Assign to Closest Mean



Assign to Closest Mean Again



Properties of k-Means

- An assignment of examples to classes is *stable* if running both the M step and the E step does not change the assignment.
- This algorithm will converge to a stable local minimum.
- It is not guaranteed to converge to a global minimum.
- It is sensitive to the relative scale of the dimensions.
- Increasing k can always decrease error until k is the number of different examples.

Soft k-Means

- To illustrate soft clustering, consider a "soft" k-means algorithm.
- E-Step: For each example e, calculate probability distribution $P(\text{class } i \mid e)$

$$P(c_i \mid e) \propto \exp\{-\|u_i - e\|^2\}$$

- M-Step: For each class i, determine mean probabilistically.

$$u_i = \frac{\sum_{e \in E} P(c_i \mid e) * e}{\sum_{e \in E} P(c_i \mid e)}$$

Soft k-Means Example

e	$P_0(C_x \mid e)$	$P_1(C_x \mid e)$	$P_2(C_x \mid e)$
$(0.7, 5.1)$	0.0	0.013	0.0
$(1.5, 6.0)$	1.0	0.764	0.0
$(2.1, 4.5)$	1.0	0.004	0.0
$(2.4, 5.5)$	0.0	0.453	0.0
$(3.0, 4.4)$	0.0	0.007	0.0
$(3.5, 5.0)$	1.0	0.215	0.0
$(4.5, 1.5)$	0.0	0.000	0.0
$(5.2, 0.7)$	0.0	0.000	0.0
$(5.3, 1.8)$	0.0	0.000	0.0
$(6.2, 1.7)$	0.0	0.000	0.0
$(6.7, 2.5)$	1.0	0.000	0.0
$(8.5, 9.2)$	1.0	1.000	1.0
$(9.1, 9.7)$	1.0	1.000	1.0
$(9.5, 8.5)$	0.0	1.000	1.0

Properties of Soft Clustering

- Soft clustering often uses a parameterized probability model, e.g., means and standard deviations for normal distribution.
- Initially, assign random probabilities to the examples: prob. of class i given example e.
- The M-step updates the values of the parameters from the probabilities.
- The E-step updates the probabilities of the examples from the probability model.
- Does not guarantee global minimum.

Reinforcement Learning

What should an agent do given:

- Prior knowledge: possible states of the world
  possible actions
- Observations: current state of world
  immediate reward/punishment
- Goal: act to maximize accumulated reward
- We assume there is a sequence of experiences:

  state, action, reward, state, action, reward, ...
- At any time agent must decide whether to *explore* to gain more knowledge, or *exploit* knowledge it has already discovered

Why is reinforcement learning hard?

- What actions are responsible for a reward may have occurred a long time before the reward was received.
- The long-term effect of an action depends on what the agent will do in the future.
- The explore-exploit dilemma: at each time should the agent be greedy or inquisitive?
  - The ϵ-greedy strategy is to select what looks like the best action $1 - \epsilon$ of the time, and to select a random action ϵ of the time.

Temporal Differences

- Suppose we have a sequence of values $v_1, v_2, v_3, \ldots$
- Estimating the average with the first k values:

$$A_k = \frac{v_1 + \cdots + v_k}{k}$$

- Separating out v_k:

$$A_k = (v_1 + \cdots + v_{k-1})/k + v_k/k$$

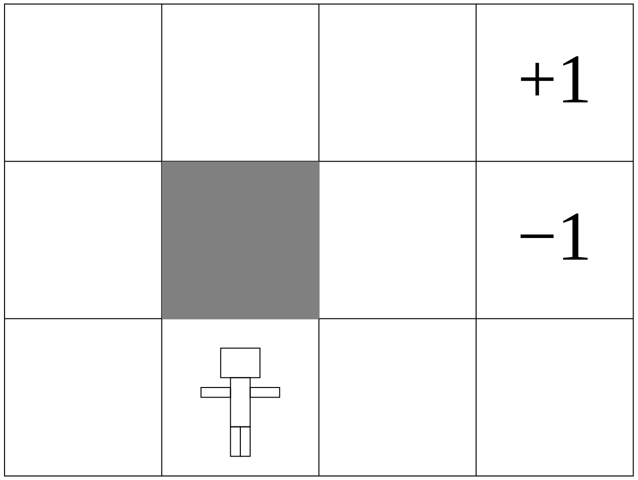
- Let $\alpha = 1/k$, then

$$A_k = (1-\alpha)A_{k-1} + \alpha v_k = A_{k-1} + \alpha(v_k - A_{k-1})$$

- The TD update is: $A \leftarrow A + \alpha(v - A)$

Reinforcement Learning Example

Suppose a robot in this environment.



One terminal square has $+1$ reward (recharge station).

One terminal square has -1 reward (falling down stairs).

An action to stay put always succeeds.

An action to move to a neighbor square,

- succeeds with probability 0.8,
- stays in the same square with prob. 0.1,
- goes to another neighbor with prob. 0.1

Should the robot try moving left or right?

Review of Q Values

- A *policy* is a function from states to actions.
- For reward sequence $r_1, r_2, \ldots$, *discounted reward* is: $V = \sum_{i=1}^{\infty} \gamma^{i-1} r_i$ (discount $= \gamma$)
- $V(s)$ is expected value of state s.
- $Q(s, a)$ is value of action a from s.
- For optimal policy:

  $V(s) = \max_a Q(s, a)$ (value of best action)

  $Q(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V(s')) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_a' Q(s', a'))$
- Learn optimal policy by learning Q values. Use each experience s, a, r, s' to update $Q[s, a]$.

Q-Learning

Procedure Q-learning$(S, A, \gamma, \alpha, \epsilon)$
  Inputs: states, actions
      discount, step size, exploration factor
  Initialize $Q[S, A]$ to zeros
  Repeat for multiple episodes:
    $s \leftarrow$ initial state
    Repeat until end of episode:
      Select action a using ϵ-greedy strategy
      Do action a. Observe reward r and state s'
      $Q[s, a] \leftarrow Q[s, a] +$
        $\alpha \left(r + \gamma \max_{a'} Q[s', a'] - Q[s, a]\right)$
      $s \leftarrow s'$
  return Q

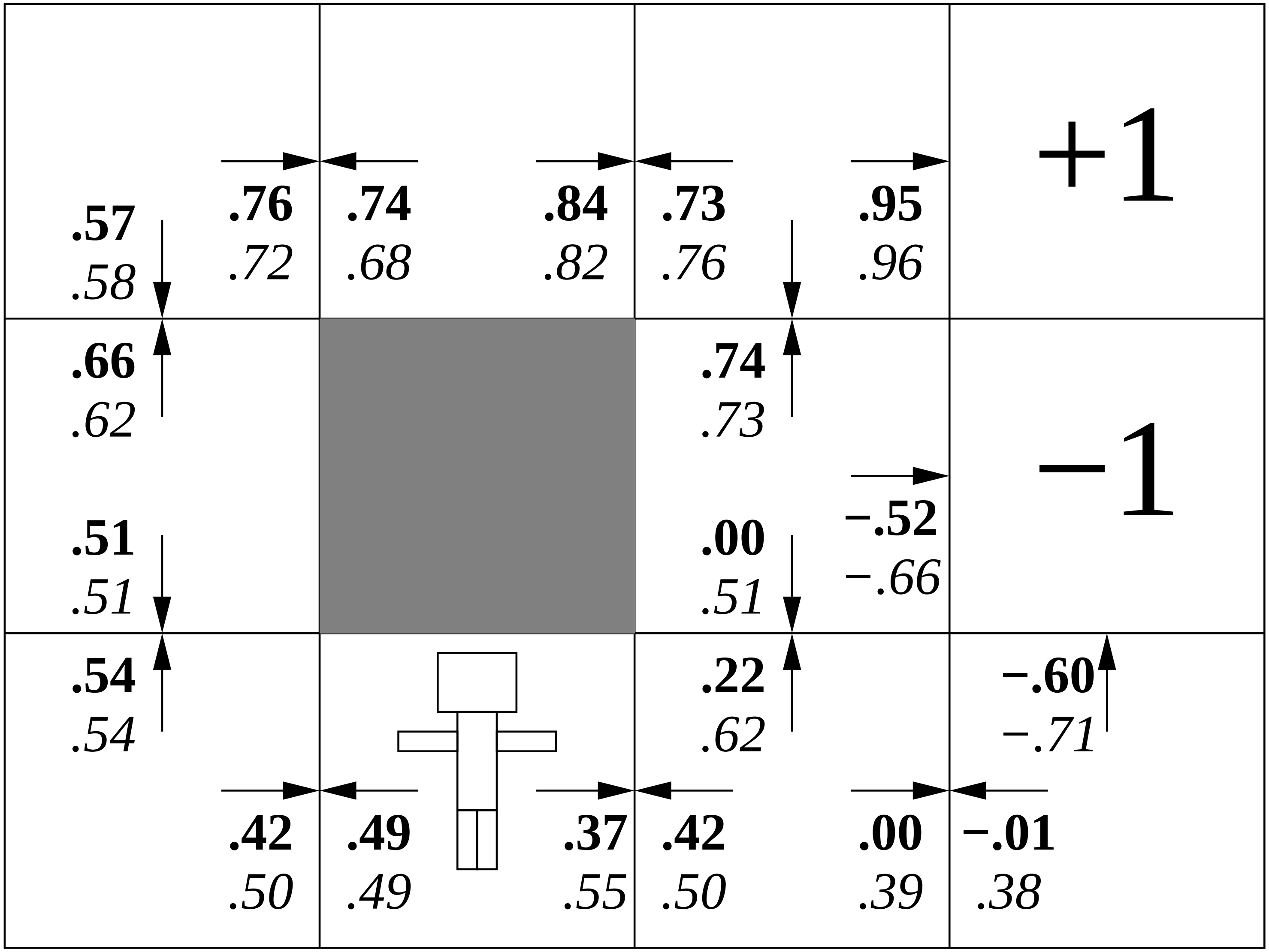
Q-Learning Update

Unpacking the Q-Learning update:

$$Q[s,a] \leftarrow Q[s,a] + \alpha\left(r + \gamma \max_{a'} Q[s',a'] - Q[s,a]\right)$$

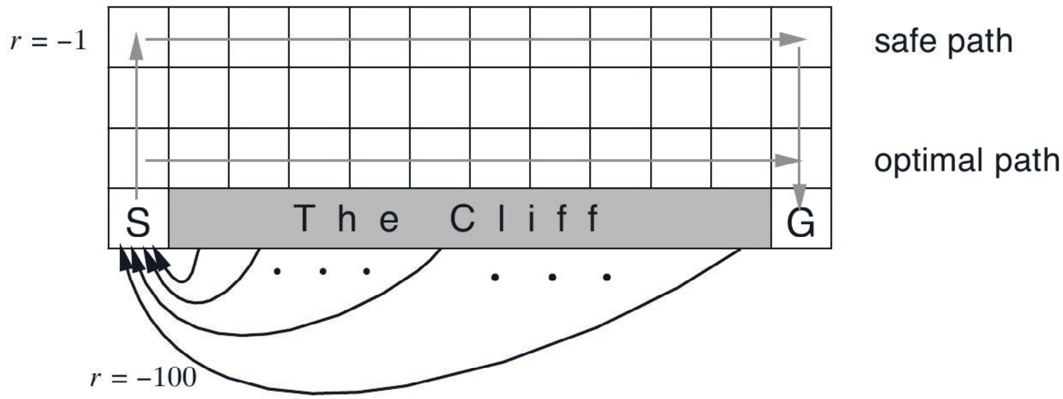
- $Q[s,a]$: Value of doing action a from state s.
- r, s': reward received and next state
- α, γ: learning rate and discount factor
- $\max_{a'} Q[s',a']$: With current Q values, the value of the optimal action from state s'.
- $r + \gamma \max_{a'} Q[s',a']$: With current Q values, the discounted reward to be averaged in.

Robot Q-Learner, $\gamma = 0.9$, $\alpha = \epsilon = 0.1$



Problems with Q-Learning

- Q-learning does off-policy learning: learns value of optimal policy, but does not follow it.
- This is bad if exploration is dangerous. Below, Q-learning walks off the cliff too much.

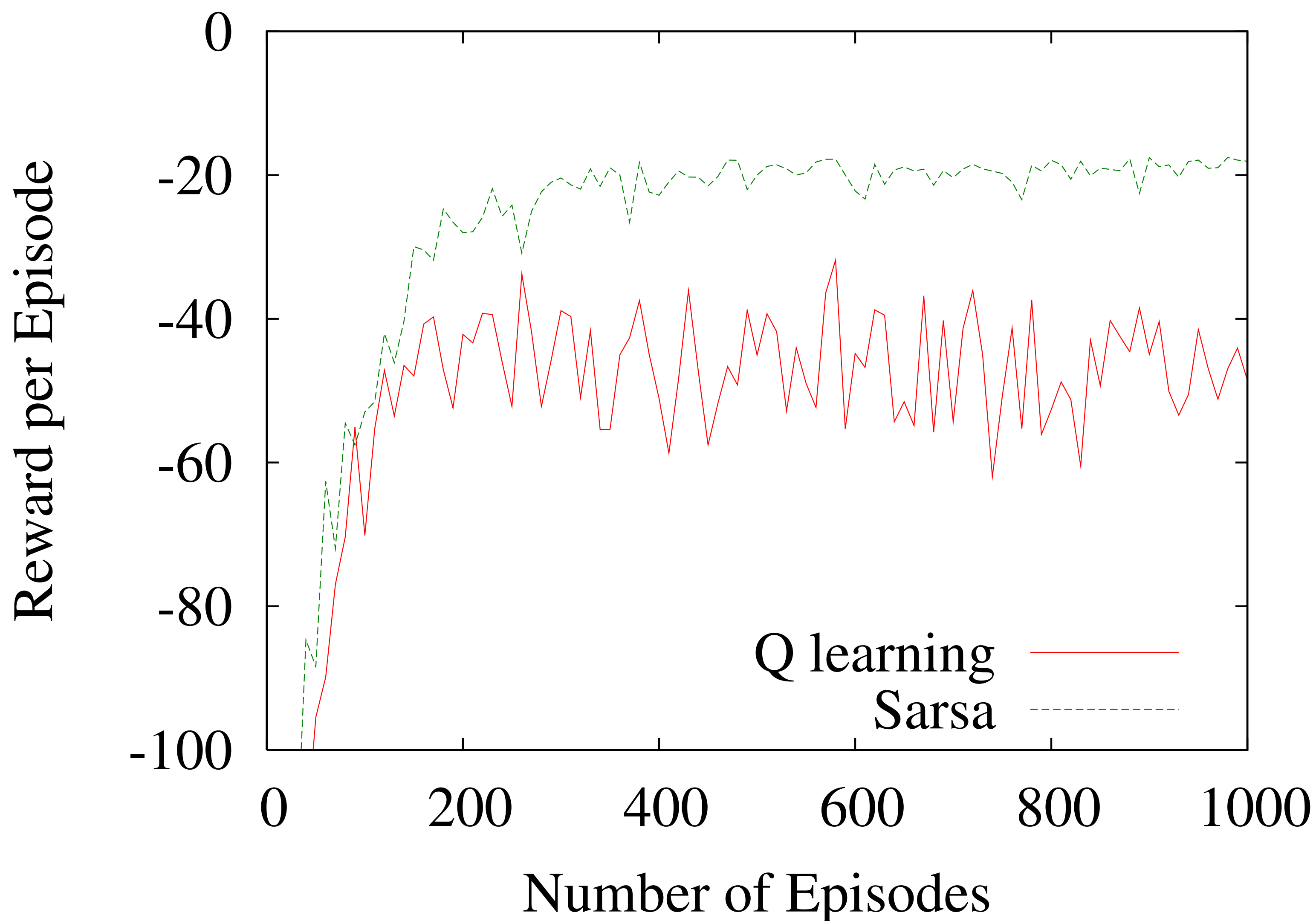


- On-policy learning learns the value of the policy being followed.
- SARSA uses the experience s, a, r, s', a' to update $Q[s, a]$.

SARSA Procedure

Procedure SARSA$(S, A, \gamma, \alpha, \epsilon)$
  Initialize $Q[S, A]$ to zeros
  Repeat for multiple episodes:
    $s \leftarrow$ initial state
    Select action a using ϵ-greedy strategy
    Repeat until end of episode:
      Do action a. Observe reward r and state s'
      Select action a' for s' using ϵ-greedy
      $Q[s, a] \leftarrow Q[s, a] + \alpha\left(r + \gamma Q[s', a'] - Q[s, a]\right)$
      $s \leftarrow s'$ and $a \leftarrow a'$
  return Q

SARSA on the Cliff



Reinforcement Learning with Features

- Often, we want to reason in terms of features.
- Want to take of advantage of similarities between states.
- Each assignment to the features is a state.
- Idea: Express Q as a function of the features. Features encode state and action.

$$(s, a) = (x_1, x_2, x_3, \ldots)$$

$$Q(s, a) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots$$

$$\delta = r + \gamma Q(s', a') - Q(s, a)$$

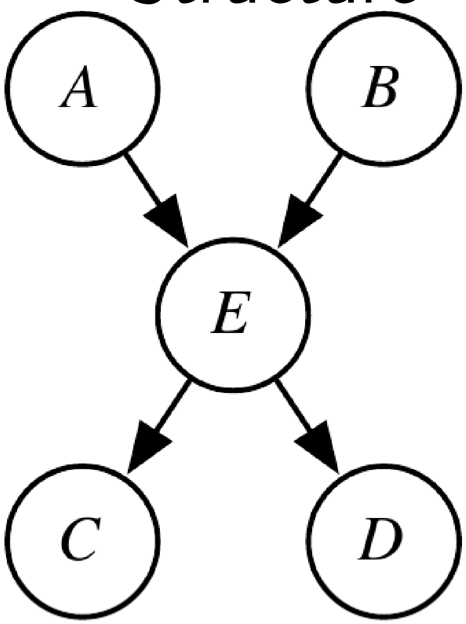
$$w_i \leftarrow w_i + \alpha \delta x_i$$

Learning Bayesian Networks

Learn each probability separately, if you:

- ☐ know the structure of the network
- ☐ observe all the variables
- ☐ have many examples
- ☐ have no missing data

Structure



Data

A	B	C	D	E
t	f	t	t	f
f	t	t	t	t
t	t	f	t	f
		$\dots$		

Probababilities

$P(A)$
$P(B)$
$P(E|A,B)$
$P(C|E)$
$P(D|E)$

Learning Conditional Probabilities

- Use counts for each conditional probability. For example:

  $$P(E = t \mid A = t \wedge B = f) = \frac{\text{count}(E = t \wedge A = t \wedge B = f) + c_1}{\text{count}(A = t \wedge B = f) + c}$$

  c_1 and c is prior (expert) knowledge ($c_1 \leq c$).
- When there are few examples or many parents to a node, there might be little data for probability estimates:
  - Use supervised learning or noisy ORs/ANDs.

Unobserved Variables

- What if we had no observations of E?
- Use EM algorithm with probabilistic inference.
  - Randomly assign values to probability tables that include E.
  - Repeat:
  - E-step: Calculate $P(E \mid e)$ for each example e.
  - M-step: Update probabability tables using counts of $P(E \mid e)$.

Learning Bayesian Network Structures

$$P(M \mid D) = \frac{P(D \mid M) * P(M)}{P(D)}$$

$$\log P(M \mid D) \propto \log P(D \mid M) + \log P(M)$$

- M is a Bayesian network and D is the data.
- Assume all variables are observed.
- A bigger network can have higher $P(M \mid D)$.
- $P(M)$ can help control the size. (e.g., using the description length).
- You can search over network structure looking for the most likely model.

Algorithm I

- Search over total orderings of variables.
- For each total ordering $X_1, \ldots, X_n$ use supervised learning to learn $P(X_i | X_1 \ldots X_{i-1})$.

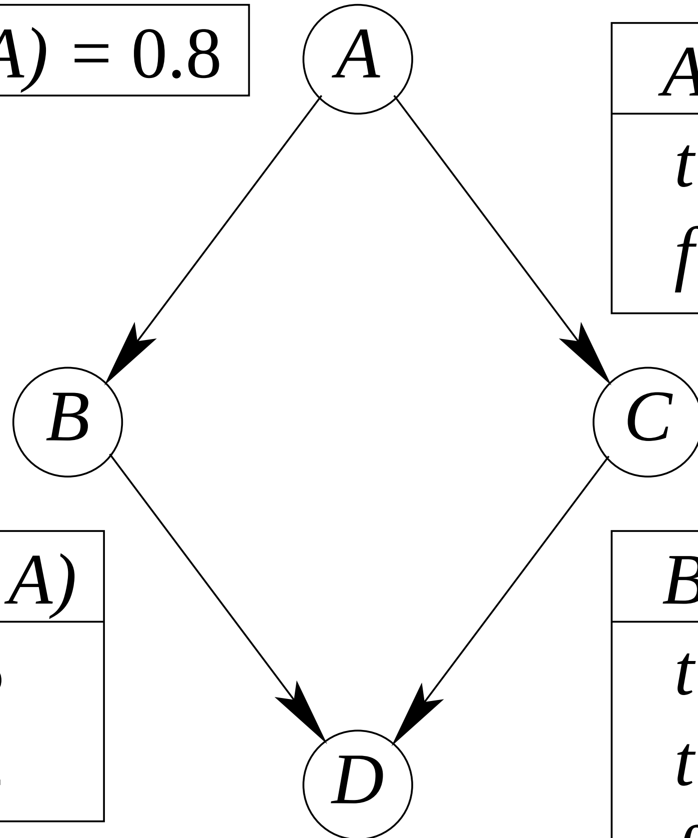
- Return the network model found with minimum: $-\log P(D \mid M) - \log P(M)$
  - $\log P(D \mid M)$ can be obtained by calculation.
  - Can approximate $\log P(M) \approx -m \log(d+1)$, where $m = \#$ of parameters in M and $d = \#$ of examples.

Algorithm II

- Learn a tree-structured Bayesian network.
- Compute correlations between all pairs of variables.
- Do maximum spanning tree maximizing absolute values of correlations.
- Pick a variable to be root of the tree, then fill in probabilities using data.

Example: Original Bayesian Network

$P(A) = 0.8$



A	$P(C \mid A)$
t	0.2
f	0.5

A	$P(B \mid A)$
t	0.5
f	0.2

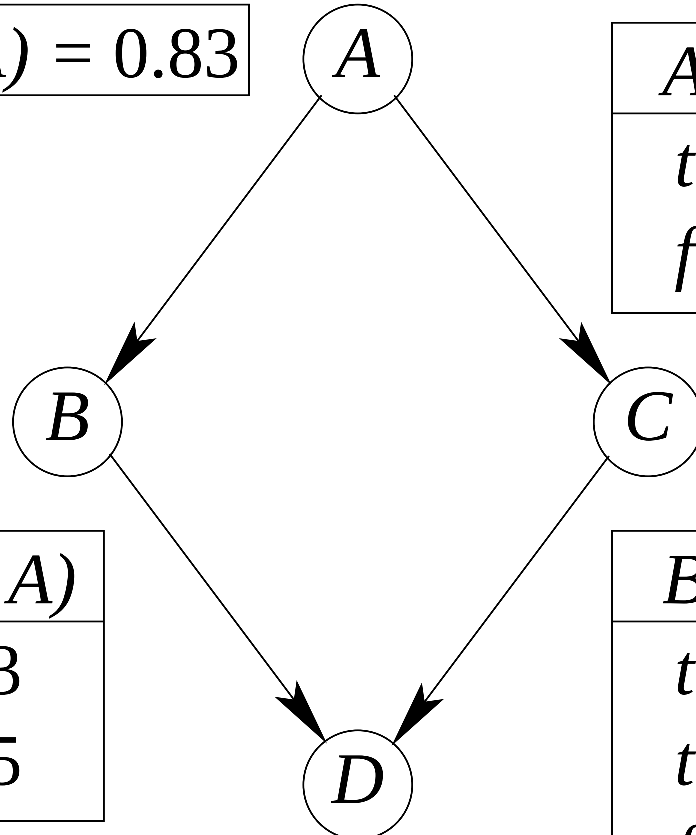
B	C	$P(D \mid B,C)$
t	t	0.5
t	f	0.8
f	t	0.2
f	f	0.4

Example: 1000 Examples

A	B	C	D	Count	A	B	C	D	Count
T	T	T	T	39	F	T	T	T	6
T	T	T	F	37	F	T	T	F	5
T	T	F	T	300	F	T	F	T	20
T	T	F	F	69	F	T	F	F	10
T	F	T	T	15	F	F	T	T	15
T	F	T	F	67	F	F	T	F	46
T	F	F	T	117	F	F	F	T	27
T	F	F	F	189	F	F	F	F	38

Example: Learn Probabilities

$P(A) = 0.83$



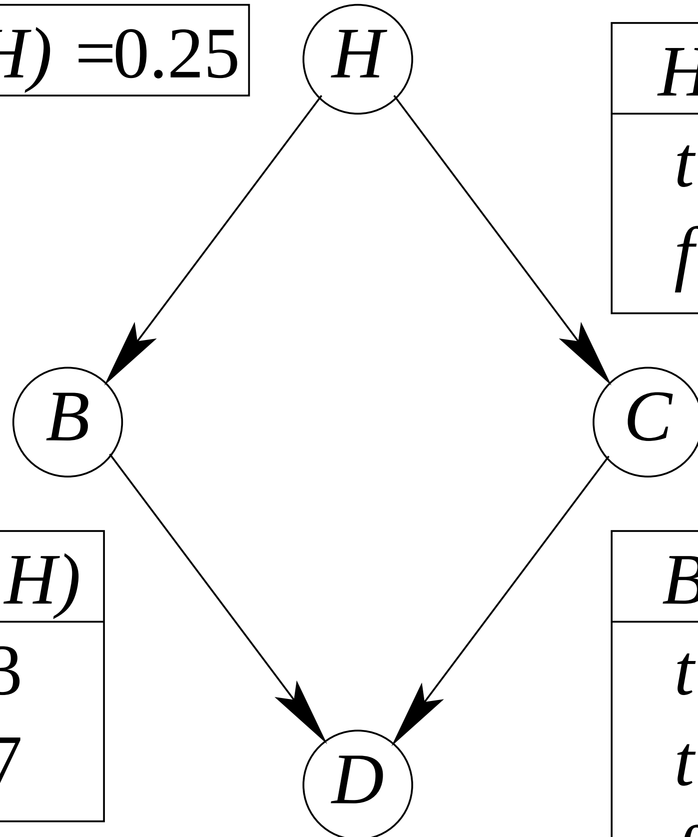
A	$P(C \mid A)$
t	0.19
f	0.43

A	$P(B \mid A)$
t	0.53
f	0.25

B	C	$P(D \mid B,C)$
t	t	0.52
t	f	0.80
f	t	0.21
f	f	0.39

Example: EM for Hidden Variable

$P(H) = 0.25$



H	$P(C \mid H)$
t	0.02
f	0.30

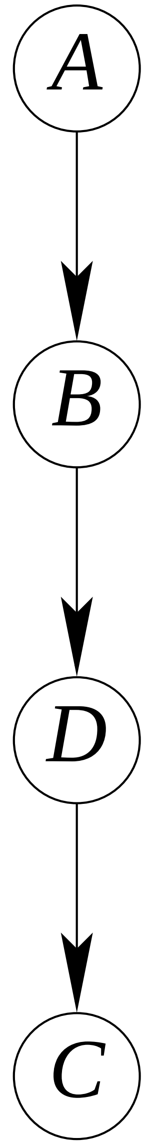
H	$P(B \mid H)$
t	0.83
f	0.37

B	C	$P(D \mid B,C)$
t	t	0.5
t	f	0.8
f	t	0.2
f	f	0.4

Example: Learn Structure

Correlation Table

	B	C	D
A	$.22$	$-.21$	$.11$
B		$-.12$	$.41$
C			$-.23$



$P(A) = 0.83$

A	$P(B \mid A)$
t	0.53
f	0.25

B	$P(D \mid B)$
t	0.75
f	0.34

D	$P(C \mid D)$
t	0.14
f	0.34