10-301/601: Introduction to Machine Learning Lecture 29 – Exam 3 Review

Henry Chai

8/9/22

Front Matter

- Announcements:
  - Exam 3 on 8/12, this Friday!
    - Exam review recitation on 8/10 (tomorrow)
    - Please show up to PH 100 (in-person) at 3:50 PM as the exam will begin promptly at 4 PM

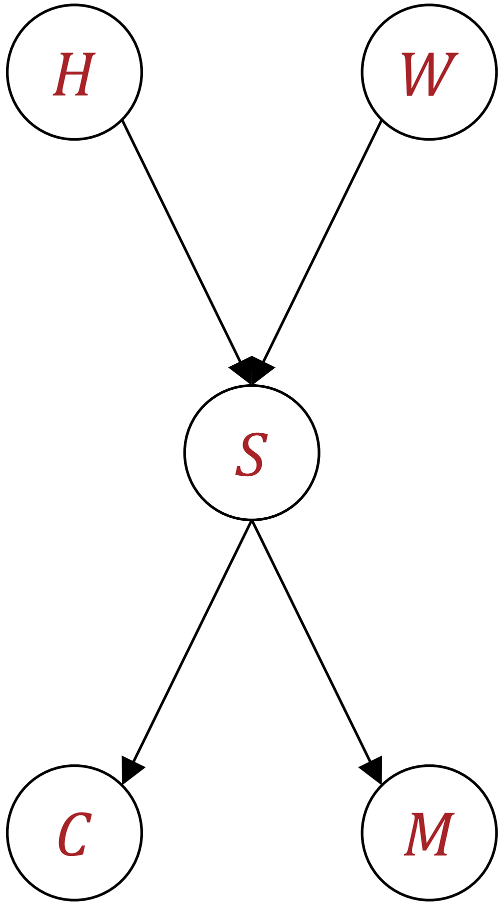
Recall: Naïve Bayes Assumption

- ***Assume*** features are conditionally independent given the label:

$$P(X|Y) = \prod_{d=1}^{D} P(X_d|Y)$$

- Pros:
  - Significantly reduces computational complexity
  - Also reduces model complexity, combats overfitting
- Cons:
  - Is a strong, often illogical assumption
    - We'll see a relaxed version of this ~~later in the semester~~ today when we discuss Bayesian networks

Constructing a Network



- Directed acyclic graph where edges indicate conditional dependency
- A variable is conditionally independent of all its non-descendants (i.e., upstream variables) given its parents
- $P(H, W, S, C, P) = P(H)P(W)P(S|H, W) P(C|S)P(M|S)$

Bayesian Networks: Outline

- How can we learn a Bayesian network?
  - Learning the graph structure
  - Learning the conditional probabilities
- What inference questions can we answer with a Bayesian network?
  - Computing (or estimating) marginal (conditional) probabilities
  - Implied (conditional) independencies

Practice Problem: Bayesian Networks

We use the following Bayesian network to model the relationship between studying (S), being well-rested (R), doing well on the exam (E), and getting an A grade (A). All nodes are binary, i.e., $R, S, E, A \in \{0, 1\}$.

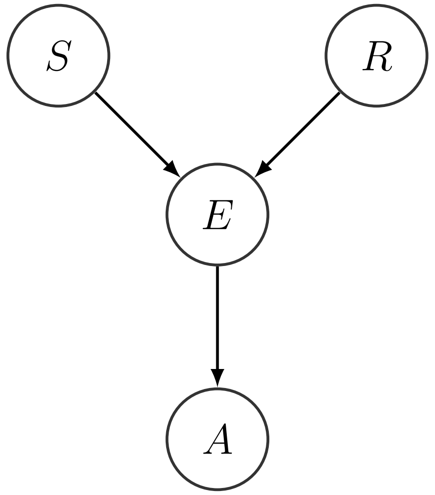
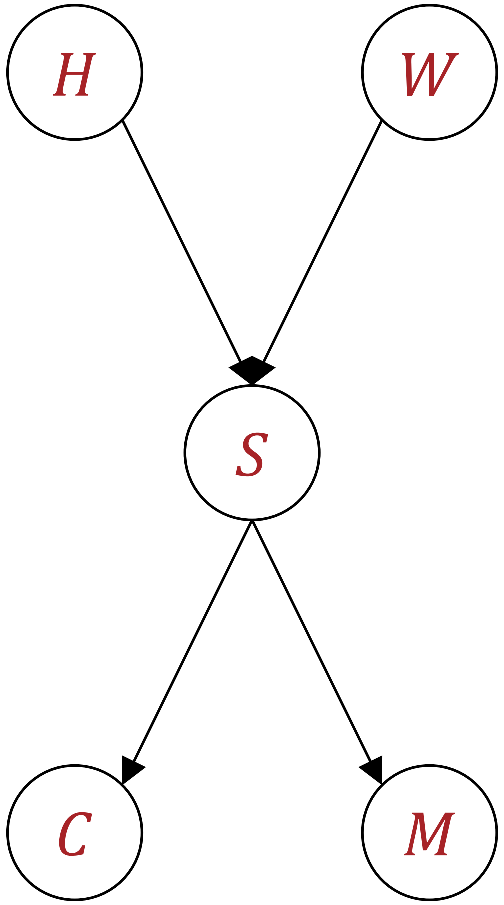


Figure 5: Directed graphical model for problem 5.

- How many parameters are needed to fully specify this Bayesian network?

Learning the Parameters (Fully-observed)



- $\mathcal{D} = \left\{\left(H^{(n)}, W^{(n)}, S^{(n)}, C^{(n)}, M^{(n)}\right)\right\}_{n=1}^{N}$
- Set parameters via MLE

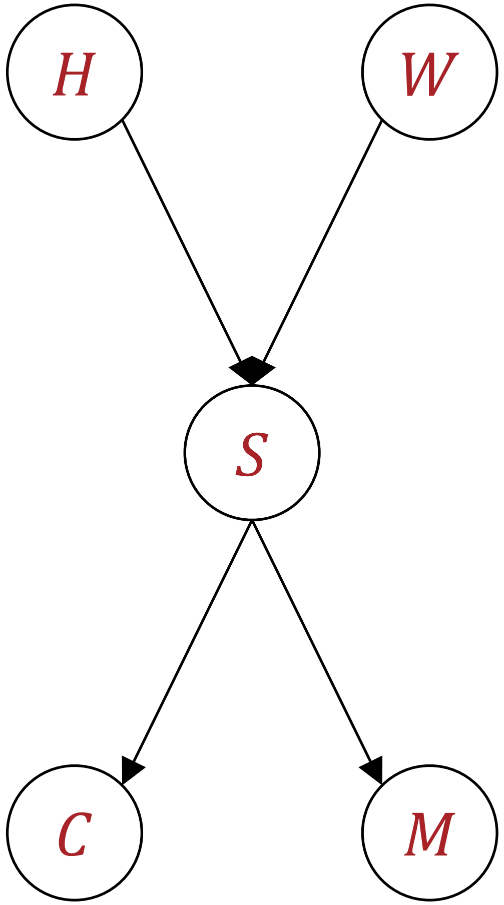
$$P(H = 1) = \frac{N_{H=1}}{N}$$

$$\vdots$$

$$P(S = 1 | H = 0, W = 1) = \frac{N_{S=1,H=0,W=1}}{N_{H=0,W=1}}$$

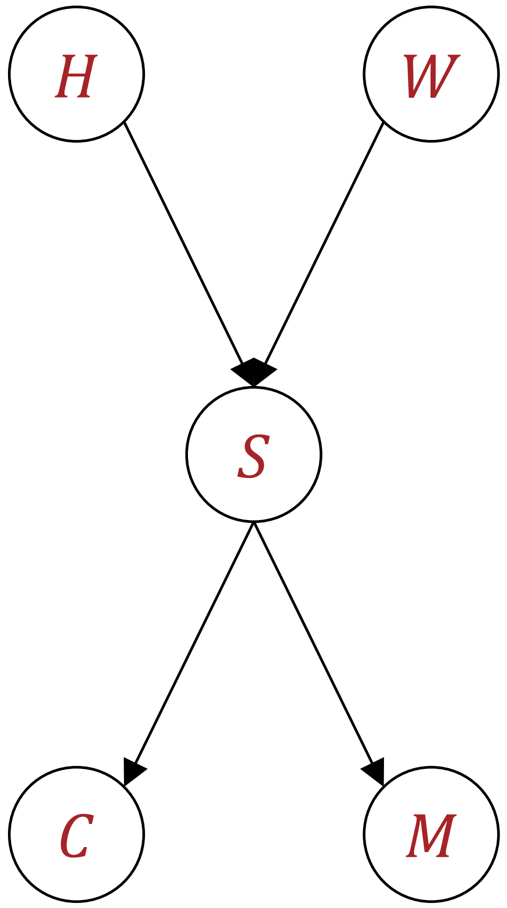
$$\vdots$$

Computing Joint Probabilities is easy



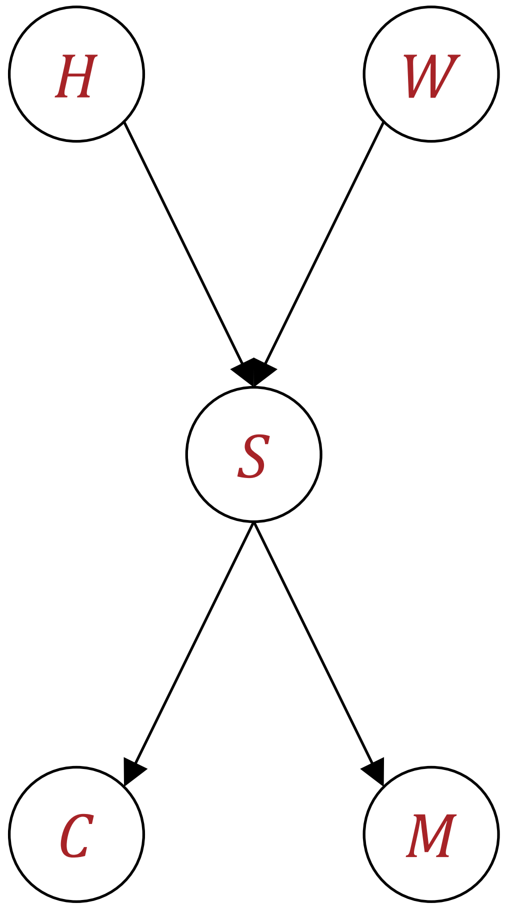
$$P(H=1, W=0, S=1, C=1, M=0)$$
$$=$$
$$P(H=1) *$$
$$\left(1 - P(W=1)\right) *$$
$$P(S=1 \mid H=1, W=0) *$$
$$P(C=1 \mid S=1) *$$
$$\left(1 - P(M=1 \mid S=1)\right)$$

Computing Marginal Probabilities…



- Computing arbitrary marginal (conditional) distributions requires summing over exponentially many possible combinations of the unobserved variables
- Computation can be improved by storing and reusing calculated values (dynamic programming)
  - Still exponential in the worst case

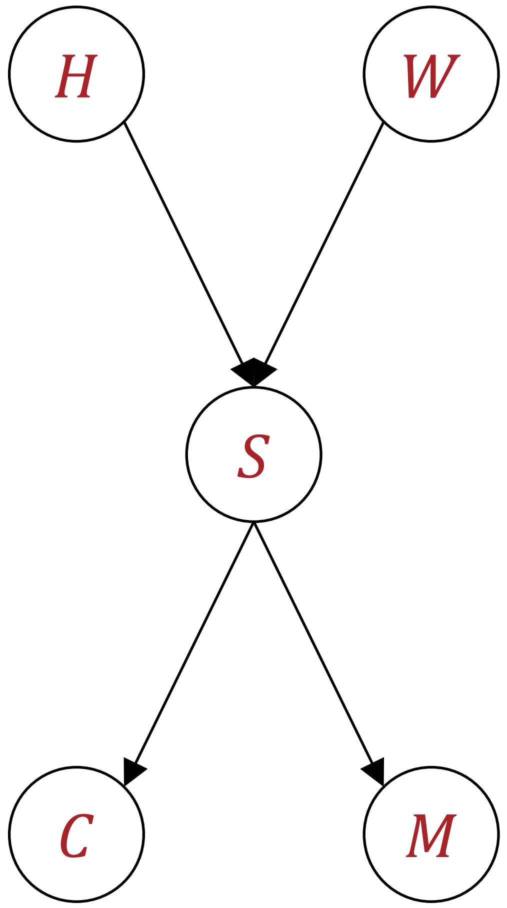
Sampling for Bayesian Networks



- Sampling from a Bayesian network is easy!
  1. Sample all free variables (H and W)
  2. Sample any variable whose parents have already been sampled
  3. Stop once all variables have been sampled

$$P(S=1) \approx \frac{\#\text{ of samples w/ } S=1}{\#\text{ of samples}}$$

Conditional Independence


$$\begin{tikzcd} H \arrow[dr] & & W \arrow[dl] \\ & S \arrow[dl] \arrow[dr] & \\ C & & M \end{tikzcd}$$

- X and Y are conditionally independent given Z ($X \perp Y \mid Z$) if
$$P(X,Y|Z) = P(X|Z)P(Y|Z)$$
- In a Bayesian network, each variable is conditionally independent of its ***non-descendants*** given its parents
  - H and M are not independent but they are conditionally independent given S
- What other conditional independencies does a Bayesian network imply?

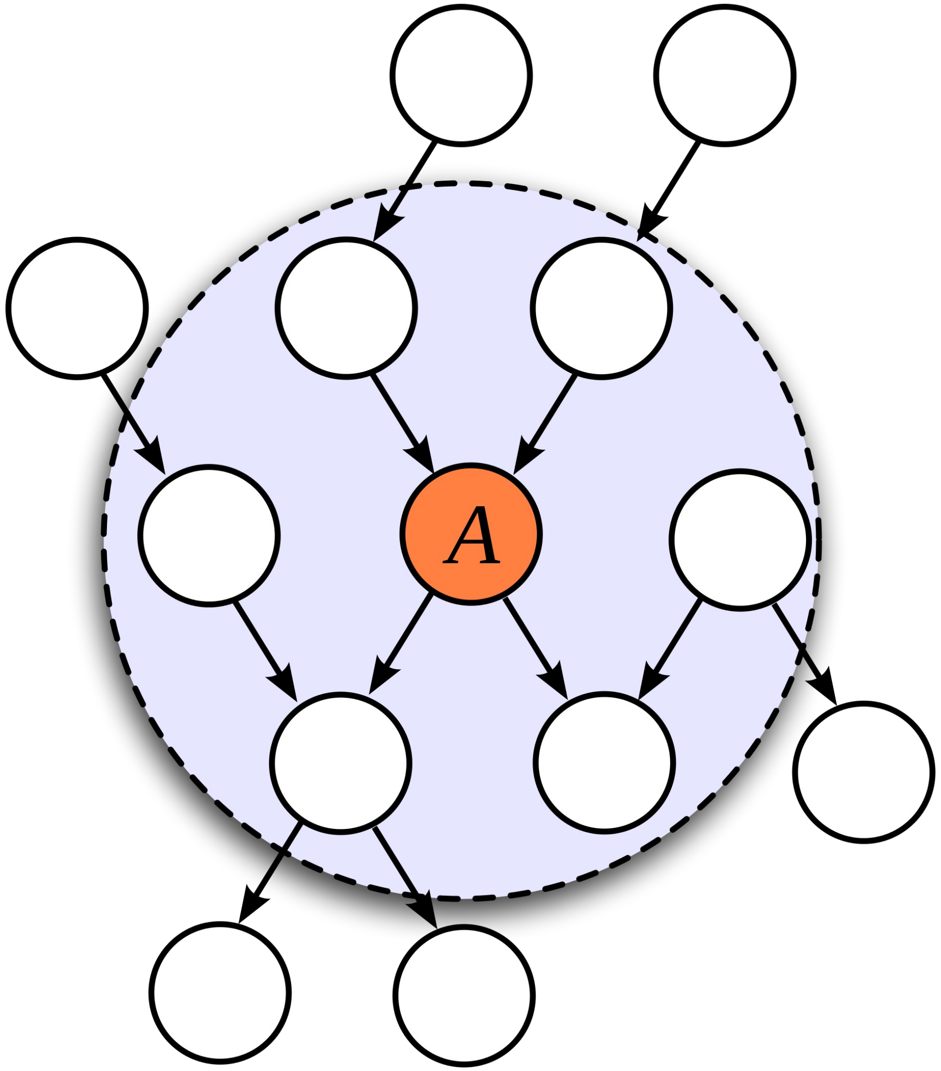
Markov Blanket

- Let $\mathcal{S}$ be the set of all random variables in a Bayesian network
- A *Markov blanket* of $A \in \mathcal{S}$ is any set $B \subseteq \mathcal{S}$ s.t.

$$A \perp \mathcal{S} \backslash B \mid B$$

  - Contains all the useful information about A
- Trivially, $\mathcal{S}$ is always a Markov blanket for any random variable in $\mathcal{S}$

Markov Boundary

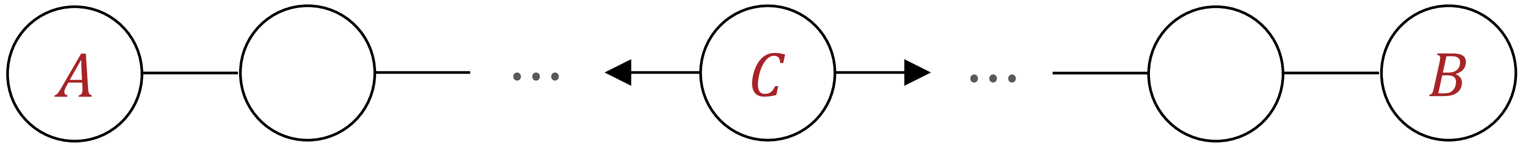


- Let $\mathcal{S}$ be the set of all random variables in a Bayesian network
- The *Markov boundary* of A is the smallest possible Markov blanket of A
- The Markov boundary consists of a variable's children, parents and co-parents (the other parents of its children)

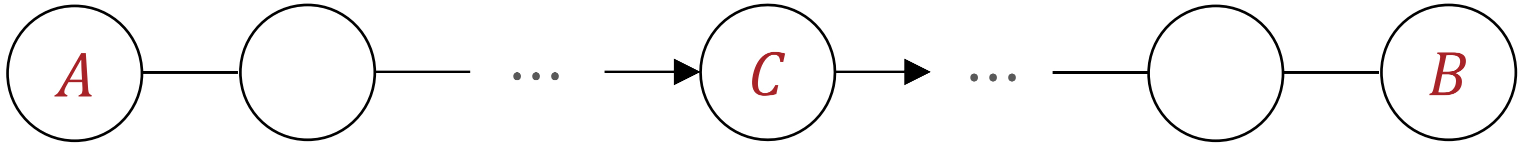
Source: https://en.wikipedia.org/wiki/Markov_blanket#/media/File:Diagram_of_a_Markov_blanket.svg

D-separation

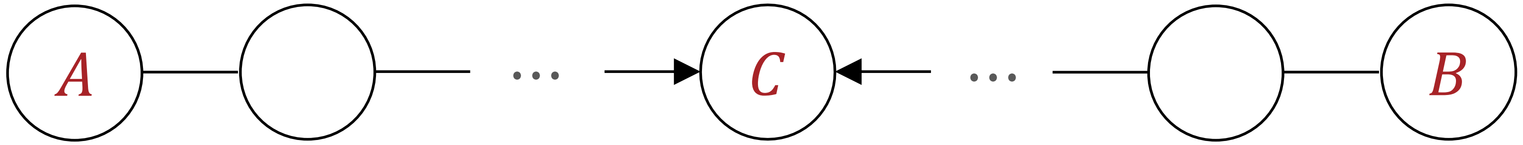
- Random variables A and B are *d-separated* given evidence variables Z if $A \perp B \mid Z$
- Definition 1: A and B are d-separated given Z iff every *undirected* path between A and B is *blocked* by Z
- An undirected path between A and B is blocked by Z if
  1. $\exists$ a "common parent" variable C on the path and $C \in Z$



  2. $\exists$ a "cascade" variable C on the path and $C \in Z$



  3. $\exists$ a "collider" variable C on the path and $\{C, \text{descendents}(C)\} \notin Z$



D-separation

- Random variables A and B are *d-separated* given evidence variables Z if $A \perp B \mid Z$
- Definition 2: A and B are d-separated given Z iff $\nexists$ a path between A and B in the undirected ancestral moral graph with Z removed
  1. Keep only A, B, Z and their ancestors (ancestral graph)
  2. Add edges between all co-parents (moral graph)
  3. Undirected: replace directed edges with undirected ones
  4. Delete Z and check if A and B are connected
- Example: $A \perp B \mid \{D, E\}$?

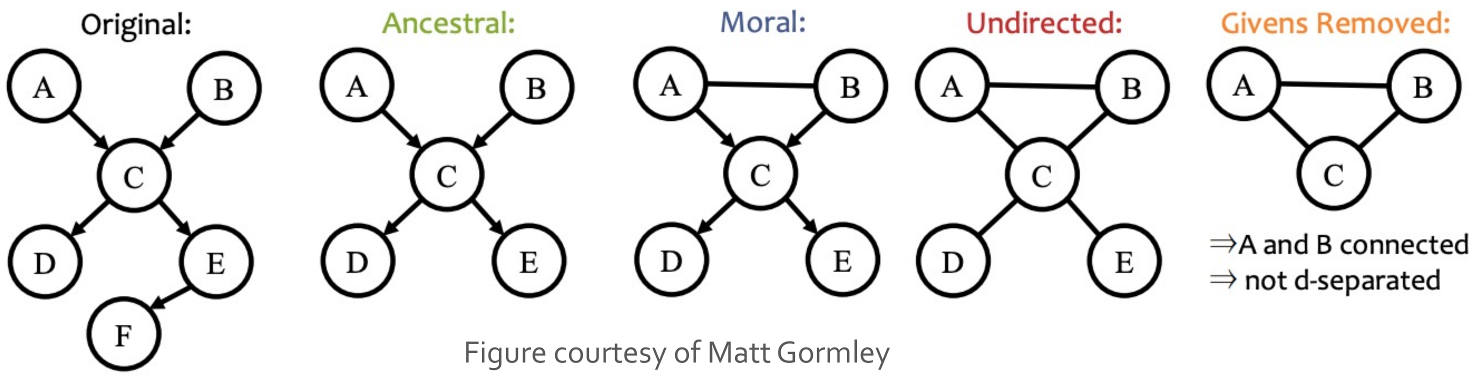


Figure courtesy of Matt Gormley

Practice Problem: Bayesian Networks

We use the following Bayesian network to model the relationship between studying (S), being well-rested (R), doing well on the exam (E), and getting an A grade (A). All nodes are binary, i.e., $R, S, E, A \in \{0, 1\}$.

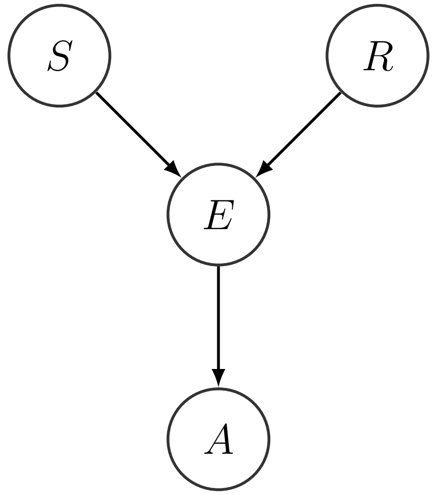


Figure 5: Directed graphical model for problem 5.

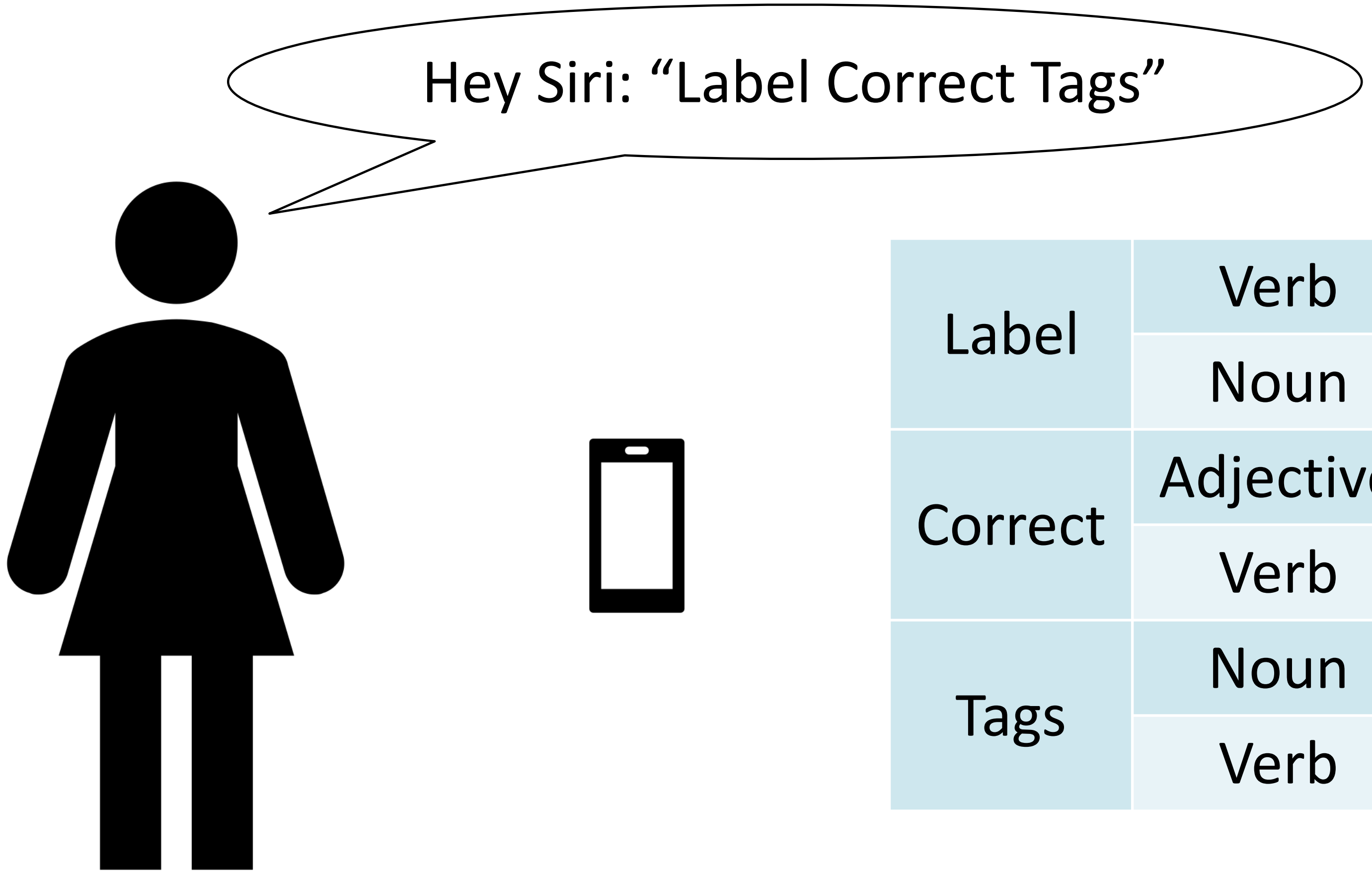
- Are S and R independent? Are S and R conditionally independent given E?

Shortcomings of Bayesian Networks

- Graph structure must be acyclic
- Cannot encode temporal/sequential relationships

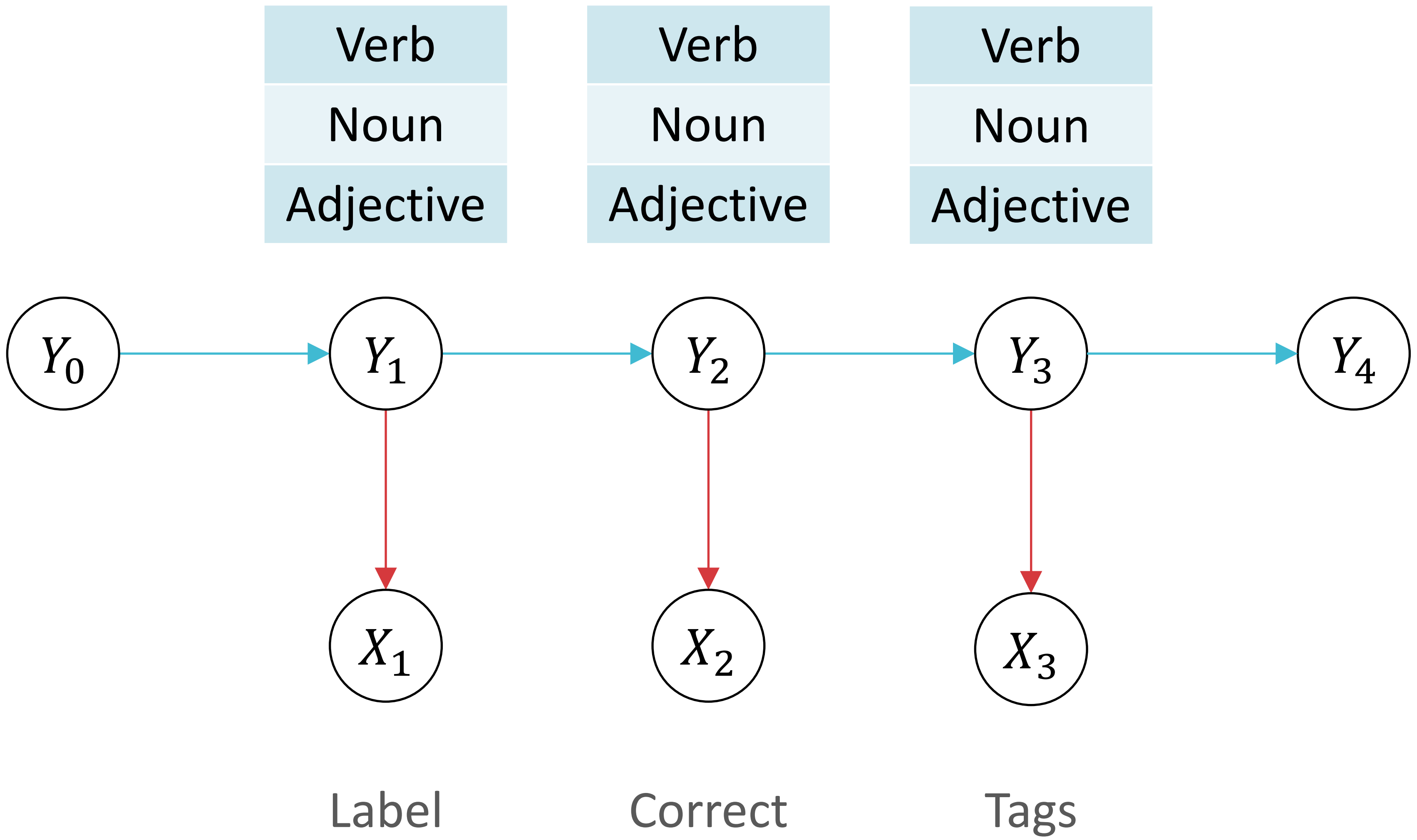
- We'll address these (related) problems ~~next~~ today with hidden Markov models

Part-of-Speech (PoS) Tagging: Example



Label	Verb
	Noun
Correct	Adjective
	Verb
Tags	Noun
	Verb

Hidden Markov Models for PoS Tagging



Hidden Markov Models

- Two types of variables: observations (observed) and states (hidden or latent)
  - Set of states usually pre-specified via domain expertise/prior knowledge: $\{s_1, \dots, s_M\}$
  - Emission model:
    - Current observation is conditionally independent of all other variables given the current state: $P(X_t | Y_t)$
  - Transition model (Markov assumption):
    - Current state is conditionally independent of all earlier states given the previous state: $P(Y_t | Y_{t-1}, \dots, Y_0) = P(Y_t | Y_{t-1})$

Hidden Markov Models vs. Bayesian Networks

- Two types of variables: observations (observed) and states (hidden or latent)
  - Set of states usually pre-specified via domain expertise/prior knowledge: $\{s_1, \dots, s_M\}$
  - Emission & transition models are fixed over time steps

$$P(X_t | Y_t = s_j) = P(X_{t'} | Y_{t'} = s_j) \; \forall \, t, t'$$

$$P(Y_t | Y_{t-1} = s_j) = P(Y_{t'} | Y_{t'-1} = s_j) \; \forall \, t, t'$$

    - Parameter reuse makes learning efficient
    - Can handle sequences of varying lengths

Hidden Markov Models: Outline

- How can we learn the conditional probabilities used by a hidden Markov model?
- What inference questions can we answer with a hidden Markov model?
  1. Computing the distribution of a single state (or a sequence of states) given a sequence of observations
  2. Finding the most-probable sequence of states given a sequence of observations
  3. Computing the probability of a sequence of observations

Practice Problem: HMMs

1. Given the POS tagging data shown, what are the parameter values learned by an HMM?

Verb	Noun	Verb
see	spot	run

Verb	Noun	Verb
run	spot	run

Adj.	Adj.	Noun
funny	funny	spot

Learning the Parameters (Fully-observed)

- $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\right)\right\}_{n=1}^{N}$
- Set the parameters via MLE

	s_1	$\cdots$	s_M
o_1	a_{11}	$\cdots$	a_{1M}
o_2	a_{21}	$\cdots$	a_{2M}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
o_C	a_{C1}	$\cdots$	a_{CM}

Emission matrix, A

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} N_{X_t = o_i,\, Y_t = s_j}}{\sum_{t=1}^{T} N_{Y_t = s_j}}$$

	START	s_1	$\cdots$	s_M
s_1	b_{10}	b_{11}	$\cdots$	b_{1M}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_M	b_{M0}	b_{M1}	$\cdots$	b_{MM}
END	$b_{(M+1)0}$	$b_{(M+1)1}$	$\cdots$	$b_{(M+1)M}$

Transition matrix, B

$$\hat{b}_{ij} = \frac{\sum_{t=1}^{T+1} N_{Y_t = s_i, Y_{t-1} = s_j}}{\sum_{t=1}^{T+1} N_{Y_{t-1} = s_j}}$$

3 Inference Questions for HMMs

1. Marginal Computation: $P\left(Y_t = s_j \middle| \boldsymbol{x}^{(n)}\right)$ (or $P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$)

$$P\left(Y \middle| \boldsymbol{x}^{(n)}\right) = \frac{P\left(\boldsymbol{x}^{(n)} | Y\right) P(Y)}{P\left(\boldsymbol{x}^{(n)}\right)} = \frac{\prod_{t=1}^{T} P\left(\boldsymbol{x}_t^{(n)} \middle| Y_t\right) P(Y_t | Y_{t-1})}{P\left(\boldsymbol{x}^{(n)}\right)}$$

2. Decoding: $\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$

3. Evaluation: $P\left(\boldsymbol{x}^{(n)}\right)$

$$P\left(\boldsymbol{x}^{(n)}\right) = \sum_{\mathcal{Y} \in \{\text{all possible sequences}\}} P\left(\boldsymbol{x}^{(n)} \middle| \mathcal{Y}\right) P(\mathcal{Y})$$

The Brute Force Algorithm

- Inputs: query $P\left(\boldsymbol{x}^{(n)}\right)$, emission matrix A, transition matrix B
- Initialize $p = 0$
- For $\mathcal{Y} \in \{\text{all possible sequences}\}$
    - Compute the joint probability

$$P(\boldsymbol{x}^{(n)}, \mathcal{Y}) = P(\boldsymbol{x}^{(n)} | \mathcal{Y}) P(\mathcal{Y}) = \prod_{t=1}^{T} P\left(\boldsymbol{x}_t^{(n)} \middle| \mathcal{Y}_t\right) P(\mathcal{Y}_t | \mathcal{Y}_{t-1})$$

    - $p \mathrel{+}= P\left(\boldsymbol{x}^{(n)}, \mathcal{Y}\right)$
- Return $p = P\left(\boldsymbol{x}^{(n)}\right)$

Practice Problem: HMMs

1. Given the POS tagging data shown, what are the parameter values learned by an HMM?

Verb	Noun	Verb
see	spot	run

Verb	Noun	Verb
run	spot	run

Adj.	Adj.	Noun
funny	funny	spot

2. How many POS tag sequences of length 23 are there?

3. How does an HMM efficiently search for the most probable sequence of tags given a 23-word sentence?

3 Inference Questions for HMMs

1. Marginal Computation: $P\left(Y_t = s_j \middle| \boldsymbol{x}^{(n)}\right)$ (or $P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$)

$$P\left(Y_t = s_j \middle| \boldsymbol{x}^{(n)}\right) = \frac{P\left(Y_t = s_j, \boldsymbol{x}^{(n)}\right)}{P\left(\boldsymbol{x}^{(n)}\right)}$$

2. Decoding: $\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$

3. Evaluation: $P\left(\boldsymbol{x}^{(n)}\right)$

$$P\left(\boldsymbol{x}^{(n)}\right) = \sum_{m=1}^{M} P\left(Y_t = s_m, \boldsymbol{x}^{(n)}\right)$$

Recursive Marginals

$$P\left(Y_t = s_j, \boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_T^{(n)}\right)$$

$$= P\left(\boldsymbol{x}_{t+1}^{(n)}, \dots, \boldsymbol{x}_T^{(n)} \middle| Y_t = s_j, \boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_t^{(n)}\right) P\left(Y_t = s_j, \boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_t^{(n)}\right)$$

By conditional independence assumptions

$$= P\left(\boldsymbol{x}_{t+1}^{(n)}, \dots, \boldsymbol{x}_T^{(n)} \middle| Y_t = s_j\right) P\left(Y_t = s_j, \boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_t^{(n)}\right)$$

$$\coloneqq \beta_t(j)\alpha_t(j)$$

Can be computed recursively (forward algorithm)

Can be computed recursively (backward algorithm)

The Forward-Backward Algorithm

- Inputs: query $P\left(Y_t = s_j \,\middle|\, \boldsymbol{x}^{(n)}\right)$, emission matrix A, transition matrix B
- Initialize $\alpha_0(\text{START}) = 1$ and $\beta_{T+1}(\text{END}) = 1$
- For $\tau = 1, \dots, T$
  - For $m = 1, \dots, M$

$$\alpha_\tau(m) = P\left(\boldsymbol{x}_\tau^{(n)} | Y_\tau = s_m\right) \sum_{k=1}^{M} P(Y_\tau = s_m | Y_{\tau-1} = s_k)\alpha_{\tau-1}(k)$$

- For $\tau = T, \dots, 1$
  - For $m = 1, \dots, M$

$$\beta_\tau(m) = \sum_{k=1}^{M} \beta_{\tau+1}(k) P\left(\boldsymbol{x}_{\tau+1}^{(n)} \middle| Y_{\tau+1} = s_k\right) P(Y_{\tau+1} = s_k | Y_\tau = s_m)$$

- Return $P\left(Y_t = s_j \,\middle|\, \boldsymbol{x}^{(n)}\right) = \frac{P\left(Y_t = s_j, \boldsymbol{x}^{(n)}\right)}{P\left(\boldsymbol{x}^{(n)}\right)} = \frac{\beta_t(j)\alpha_t(j)}{\sum_{m=1}^{M} \beta_t(m)\alpha_t(m)}$

Most Probable State Sequence

$$\omega_t(j) := \max_{\mathcal{Y} \in \{\text{all possible sequences of } t-1 \text{ states}\}} P\left(\mathcal{Y}, Y_t = s_j, \boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_t^{(n)}\right)$$

$=$ the probability of the most probable sequence of t states that ends in s_j, conditioned on the first t observations

$$= \max_{m \in \{1,\dots,M\}} \omega_{t-1}(m)\, P(Y_t = s_j | Y_{t-1} = s_m) P\left(\boldsymbol{x}_t^{(n)} | Y_t = s_j\right)$$

The Viterbi Algorithm

- Inputs: observations $\boldsymbol{x}^{(n)}$, emission matrix A, transition matrix B
- Initialize $\omega_0(\text{START}) = 1$
- For $\tau = 1, \dots, T+1$
  - For $m = 1, \dots, M$

$$\omega_\tau(m) = \max_{k \in \{1,\dots,M\}} P\left(\boldsymbol{x}_\tau^{(n)} | Y_\tau = s_m\right) P(Y_\tau = s_m | Y_{\tau-1} = s_k)\omega_{\tau-1}(k)$$

$$\rho_\tau(m) = \operatorname*{argmax}_{k \in \{1,\dots,M\}} P\left(\boldsymbol{x}_\tau^{(n)} | Y_\tau = s_m\right) P(Y_\tau = s_m | Y_{\tau-1} = s_k)\omega_{\tau-1}(k)$$

- Return the most probable assignment given $\boldsymbol{x}^{(n)}$:
  - $\hat{Y}_T = \rho_{T+1}(\text{END})$
  - For $\tau = T-1, \dots, 1$
    - $\hat{Y}_\tau = \rho_{\tau+1}(\hat{Y}_{\tau+1})$

~~3~~ 4 Inference Questions for HMMs

1. Marginal Computation: $P\left(Y_t = s_j \,\middle|\, \boldsymbol{x}^{(n)}\right)$ (or $P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$)

$$P\left(Y \middle| \boldsymbol{x}^{(n)}\right) = \frac{P\left(\boldsymbol{x}^{(n)} | Y\right) P(Y)}{P\left(\boldsymbol{x}^{(n)}\right)} = \frac{\prod_{t=1}^{T} P\left(\boldsymbol{x}_t^{(n)} \middle| Y_t\right) P(Y_t | Y_{t-1})}{P\left(\boldsymbol{x}^{(n)}\right)}$$

1. Viterbi Decoding: $\hat{Y} = \underset{Y}{\operatorname{argmax}}\ P\left(Y \middle| \boldsymbol{x}^{(n)}\right)$

2. Evaluation: $P\left(\boldsymbol{x}^{(n)}\right)$

$$P\left(\boldsymbol{x}^{(n)}\right) = \sum_{\mathcal{Y} \in \{\text{all possible sequences}\}} P\left(\boldsymbol{x}^{(n)} \middle| \mathcal{Y}\right) P(\mathcal{Y})$$

3. Minimum Bayes Risk (MBR) Decoding:

$$\hat{Y} = \underset{Y}{\operatorname{argmin}}\ \mathbb{E}_{Y' \sim P_{A,B}\left(\cdot \middle| \boldsymbol{x}^{(n)}\right)}\left[\ell(Y, Y')\right]$$

Learning Paradigms

- Supervised learning - $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
  - Regression - $y^{(n)} \in \mathbb{R}$
  - Classification - $y^{(n)} \in \{1, \ldots, C\}$
- Unsupervised learning - $\mathcal{D} = \left\{\boldsymbol{x}^{(n)}\right\}_{n=1}^{N}$
  - Clustering
  - Dimensionality reduction
- Reinforcement learning - $\mathcal{D} = \left\{\boldsymbol{s}^{(n)}, \boldsymbol{a}^{(n)}, r^{(n)}\right\}_{n=1}^{N}$

Outline

- Problem formulation
  - Time discounted cumulative reward
  - Markov decision processes (MDPs)
- Algorithms:
  - Value & policy iteration (dynamic programming)
  - (Deep) Q-learning (temporal difference learning)

Practice Problem: MDPs

- In reinforcement learning, our model consists of multiple kinds of functions; for each of following functions, fill in the domains (input space) and ranges (output space). Choose from: $\mathcal{S}$ (state space), $\mathcal{A}$ (action space), $\mathbb{R}$ (set of real numbers) or a combination from any of these sets.

	Domain	Range
Transition function		
Reward function		
Policy		
Value function		
Q function		

Markov Decision Process (MDP)

- Assume the following model for our data:

1. Start in some initial state s_0
2. For time step t:
    1. Agent observes state s_t
    2. Agent takes action $a_t = \pi(s_t)$
    3. Agent receives reward $r_t \sim p(r \mid s_t, a_t)$
    4. Agent transitions to state $s_{t+1} \sim p(s' \mid s_t, a_t)$
3. Total reward is $\sum_{t=0}^{\infty} \gamma^t r_t$

- MDPs make the *Markov assumption*: the reward and next state only depend on the current state and action.

Reinforcement Learning: Objective Function

- Find a policy $\pi^* = \underset{\pi}{\operatorname{argmax}}\ V^{\pi}(s)\ \forall\, s \in \mathcal{S}$
- $V^{\pi}(s) = \mathbb{E}[$*discounted* total reward of starting in state s and executing policy π forever$]$

$$= \mathbb{E}_{p(s' \mid s, a)}[R(s_0 = s, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \cdots]$$

$$= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{p(s' \mid s, a)}[R(s_t, \pi(s_t))]$$

where $0 < \gamma < 1$ is some discount factor for future rewards

Value Function

- $V^{\pi}(s) = \mathbb{E}[$discounted total reward of starting in state s and executing policy π forever$]$

$$= \mathbb{E}[R(s_0, \pi(s_0)) + \gamma R(s_1, \pi(s_1)) + \gamma^2 R(s_2, \pi(s_2)) + \cdots \mid s_0 = s]$$

$$= R(s, \pi(s)) + \gamma \mathbb{E}[R(s_1, \pi(s_1)) + \gamma R(s_2, \pi(s_2)) + \ldots \mid s_0 = s]$$

$$= R(s, \pi(s)) + \gamma \textstyle\sum_{s_1 \in \mathcal{S}} p(s_1 \mid s, \pi(s)) \big(R(s_1, \pi(s_1)) + \gamma \mathbb{E}[R(s_2, \pi(s_2)) + \cdots \mid s_1]\big)$$

$$\underbrace{\mathrm{V}^{\pi}(\mathrm{s}) = R(s, \pi(s)) + \gamma \sum_{s_1 \in \mathcal{S}} p(s_1 \mid s, \pi(s)) V^{\pi}(s_1)}_{\text{Bellman equations}}$$

Optimality

- Optimal value function:

$$V^*(s) = \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')$$

  - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables

- Optimal policy:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \underbrace{R(s,a)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')}_{\text{(Discounted) Future reward}}$$

Fixed Point Iteration

- Iterative method for solving a system of equations
- Given some equations and initial values

$$x_1 = f_1(x_1, \ldots, x_n)$$
$$\vdots$$
$$x_n = f_n(x_1, \ldots, x_n)$$
$$x_1^{(0)}, \ldots, x_n^{(0)}$$

- While not converged, do

$$x_1^{(t+1)} \leftarrow f_1\left(x_1^{(t)}, \ldots, x_n^{(t)}\right)$$
$$\vdots$$
$$x_n^{(t+1)} \leftarrow f_n\left(x_1^{(t)}, \ldots, x_n^{(t)}\right)$$

Synchronous Value Iteration

```
• Inputs: $R(s,a)$, $p(s' \mid s,a)$
• Initialize $V^{(0)}(s) = 0 \; \forall \, s \in \mathcal{S}$ (or randomly) and set $t = 0$
• While not converged, do:
    • For $s \in \mathcal{S}$
        • For $a \in \mathcal{A}$
              $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')$
        • $V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
    • $t = t + 1$
• For $s \in \mathcal{S}$
      $\pi^*(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')$
• Return $\pi^*$
```

Asynchronous Value Iteration

- Inputs: $R(s,a)$, $p(s' \mid s,a)$
- Initialize $V^{(0)}(s) = 0 \; \forall \, s \in \mathcal{S}$ (or randomly) and set $t = 0$
- While not converged, do:
    - For $s \in \mathcal{S}$
        - For $a \in \mathcal{A}$

          $$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')$$
        - $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
- For $s \in \mathcal{S}$

  $$\pi^*(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V(s')$$
- Return π^*

Policy Iteration

- Inputs: $R(s, a)$, $p(s' \mid s, a)$
- Initialize π randomly
- While not converged, do:
  - Solve the Bellman equations defined by policy π

$$V^{\pi}(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in \mathcal{S}} p\big(s' \mid s, \pi(s)\big) V^{\pi}(s')$$

  - Update π

$$\pi(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) V^{\pi}(s')$$

- Return π

Value Iteration Theory

- **Theorem 1**: Value function convergence

  V will converge to V^* if each state is "visited" infinitely often (Bertsekas, 1989)

- **Theorem 2**: Convergence criterion

  if $\max_{s \in \mathcal{S}} \left| V^{(t+1)}(s) - V^{(t)}(s) \right| < \epsilon$,

  then $\max_{s \in \mathcal{S}} \left| V^{(t+1)}(s) - V^{*}(s) \right| < \frac{2\epsilon\gamma}{1-\gamma}$ (Williams & Baird, 1993)

- **Theorem 3**: Policy convergence

  The "greedy" policy, $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s,a)$, converges to the optimal π^* in a finite number of iterations, often before the value function has converged! (Bertsekas, 1987)

$Q^*(s,a)$ w/ deterministic rewards

- $Q^*(s,a) = \mathbb{E}[$total discounted reward of taking action a in state s, assuming all future actions are optimal$]$

$$= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')$$

$$V^*(s') = \max_{a' \in \mathcal{A}} Q^*(s',a')$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[\max_{a' \in \mathcal{A}} Q^*(s',a') \right]$$

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a)$$

- Insight: if we know Q^*, we can compute an optimal policy π^*!

$Q^*(s,a)$ w/ deterministic rewards and transitions

- $Q^*(s,a) = \mathbb{E}[$total discounted reward of taking action a in state s, assuming all future actions are optimal$]$

$$= R(s,a) + \gamma V^*\big(\delta(s,a)\big)$$

- $V^*\big(\delta(s,a)\big) = \max_{a' \in \mathcal{A}} Q^*(\delta(s,a), a')$

$$Q^*(s,a) = R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(\delta(s,a), a')$$

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q^*(s,a)$$

- Insight: if we know Q^*, we can compute an optimal policy π^*!

Learning $Q^*(s, a)$ w/ deterministic rewards and transitions

Algorithm 1: Online learning (table form)

- Inputs: discount factor γ, an initial state s
- Initialize $Q(s, a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}|\times|\mathcal{A}|$ array)
- While TRUE, do
    - Take a random action a
    - Receive reward $r = R(s, a)$
    - Update the state: $s \leftarrow s'$ where $s' = \delta(s, a)$
    - Update $Q(s, a)$:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

Learning $Q^*(s,a)$ w/ deterministic rewards and transitions

Algorithm 2: ϵ-greedy online learning (table form)

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0, 1]$
- Initialize $Q(s,a) = 0 \; \forall \; s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s,a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s,a)$
  - Update the state: $s \leftarrow s'$ where $s' = \delta(s,a)$
  - Update $Q(s,a)$:
    $$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

Learning $Q^*(s, a)$ w/ deterministic rewards

Algorithm 3: ϵ-greedy online learning (table form)

- Inputs: discount factor γ, an initial state s,
  greediness parameter $\epsilon \in [0, 1]$,
  learning rate $\alpha \in [0, 1]$ ("trust parameter")
- Initialize $Q(s, a) = 0 \; \forall \, s \in \mathcal{S}, a \in \mathcal{A}$ (Q is a $|\mathcal{S}| \times |\mathcal{A}|$ array)
- While TRUE, do
  - With probability ϵ, take the greedy action
    $$a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \; Q(s, a')$$
    Otherwise, with probability $1 - \epsilon$, take a random action a
  - Receive reward $r = R(s, a)$
  - Update the state: $s \leftarrow s'$ where $s' \sim p(s' \mid s, a)$
  - Update $Q(s, a)$:

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{Current value}} + \alpha \left(\overbrace{\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{Temporal difference target}} - Q(s,a)}^{\text{Temporal difference}} \right)$$

Learning $Q^*(s, a)$: Convergence

- For Algorithm 3 (temporal difference learning), Q converges to Q^* if
  1. Every valid state-action pair is visited infinitely often
     - Q-learning is exploration-insensitive: any visitation strategy that satisfies this property will work!
  2. $0 \leq \gamma < 1$
  3. $\exists \beta$ s.t. $|R(s, a)| < \beta \ \forall \ s \in \mathcal{S}, a \in \mathcal{A}$
  4. Initial Q values are finite
  5. Learning rate α_t follows some "schedule" s.t. $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ e.g., $\alpha_t = {}^{1}/_{t+1}$

Deep Q-learning: Loss Function

- "True" loss

$$\ell(\Theta) = \underbrace{\sum_{s \in \mathcal{S}}}_{\text{1. } \mathcal{S} \text{ too big to compute this sum}} \sum_{a \in \mathcal{A}} \Big(\overbrace{Q^*(s,a)}^{\text{2. Don't know } Q^*} - Q(s,a;\Theta)\Big)^2$$

1. Use stochastic gradient descent: just consider one state-action pair in each iteration
2. Use temporal difference learning:
   - Given current parameters $\Theta^{(\mathrm{t})}$ the temporal difference target is

     $$Q^*(s,a) \approx r + \gamma \max_{a'} Q\left(s', a'; \Theta^{(t)}\right) := y$$

   - Set the parameters in the next iteration $\Theta^{(\mathrm{t+1})}$ such that $Q\left(s,a;\Theta^{(\mathrm{t+1})}\right) \approx y$

     $$\ell\left(\Theta^{(\mathrm{t})}, \Theta^{(t+1)}\right) = \left(y - Q\left(s,a;\Theta^{(\mathrm{t+1})}\right)\right)^2$$

Deep Q-learning

Algorithm 4: Online learning (parametric form)

- Inputs: discount factor γ, an initial state s_0,

  learning rate α
- Initialize parameters $\Theta^{(0)}$
- For $t = 0, 1, 2, \ \dots$
  - Gather training sample $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$
  - Update $\Theta^{(t)}$ by taking a step opposite the gradient

    $$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_{\Theta^{(t+1)}} \ell\left(\Theta^{(t)}, \Theta^{(t+1)}\right)$$

    where

    $$\nabla_{\Theta^{(t+1)}} \ell\left(\Theta^{(t)}, \Theta^{(t+1)}\right)$$
    $$= 2\left(y - Q\left(s, a; \Theta^{(t+1)}\right)\right) \nabla_{\Theta^{(t+1)}} Q\left(s, a; \Theta^{(t+1)}\right)$$

Deep Q-learning: Experience Replay

- SGD assumes i.i.d. training samples but in RL, samples are *highly* correlated
- Idea: keep a "replay memory" $\mathcal{D} = \{e_1, e_2, \dots, e_N\}$ of the N most recent experiences $e_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ (Lin, 1992)
  - Also keeps the agent from "forgetting" about recent experiences
- Alternate between:
  1. Sampling some e_i uniformly at random from $\mathcal{D}$ and applying a Q-learning update (repeat T times)
  2. Adding a new experience to $\mathcal{D}$
- Can also sample experiences from $\mathcal{D}$ according to some distribution that prioritizes experiences with high error (Schaul et al., 2016)

Clustering

- Goal: split an unlabeled data set into groups or clusters of "similar" data points
- Use cases:
  - Organizing data
  - Discovering patterns or structure
  - Preprocessing for downstream machine learning tasks
- Applications:

Clustering Algorithms

- Hierarchical
    - Top-down (divisive)
    - Bottom-up (agglomerative)
- Partitioning
    - K-means

Hierarchical Clustering

- Bottom-up (agglomerative)
  - Start with each data point in its own cluster
  - Iteratively combine the most similar clusters
  - Stop when all data points are in a single cluster

- Top-down (divisive)
  - Start with all data points in one cluster
  - Iteratively split the largest cluster into two clusters
  - Stop when all clusters are single data points

Bottom-up Hierarchical Clustering

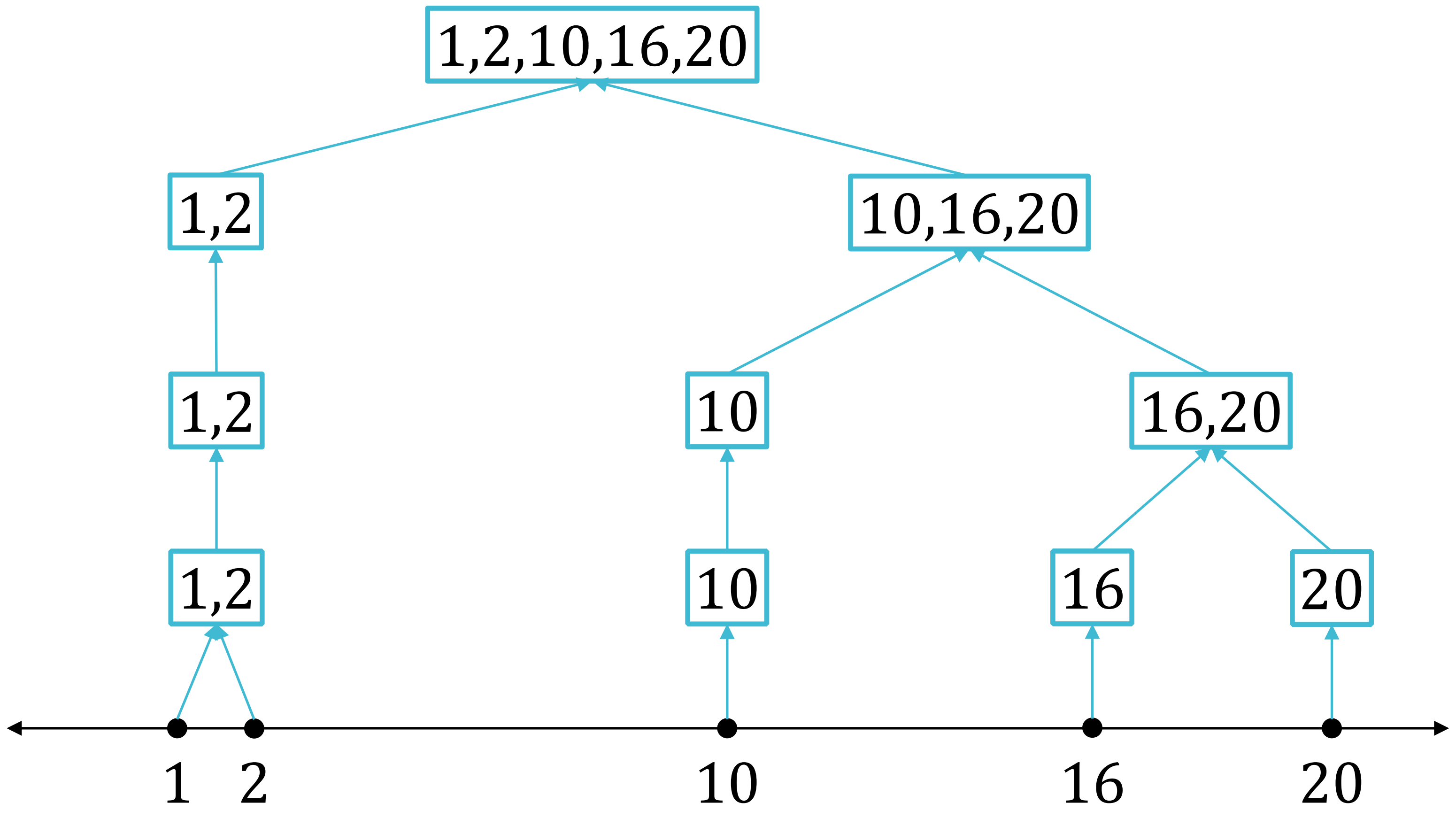
- Bottom-up (agglomerative)
  - Start with each data point in its own cluster
  - Iteratively combine the ***most similar*** clusters
  - Stop when all data points are in a single cluster
- Key question: how do we define similarity between clusters?
  - Single-linkage: consider the closest data points

$$d_{SL}\left(C_i, C_j\right) = \min_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} d(\boldsymbol{x}, \boldsymbol{y})$$

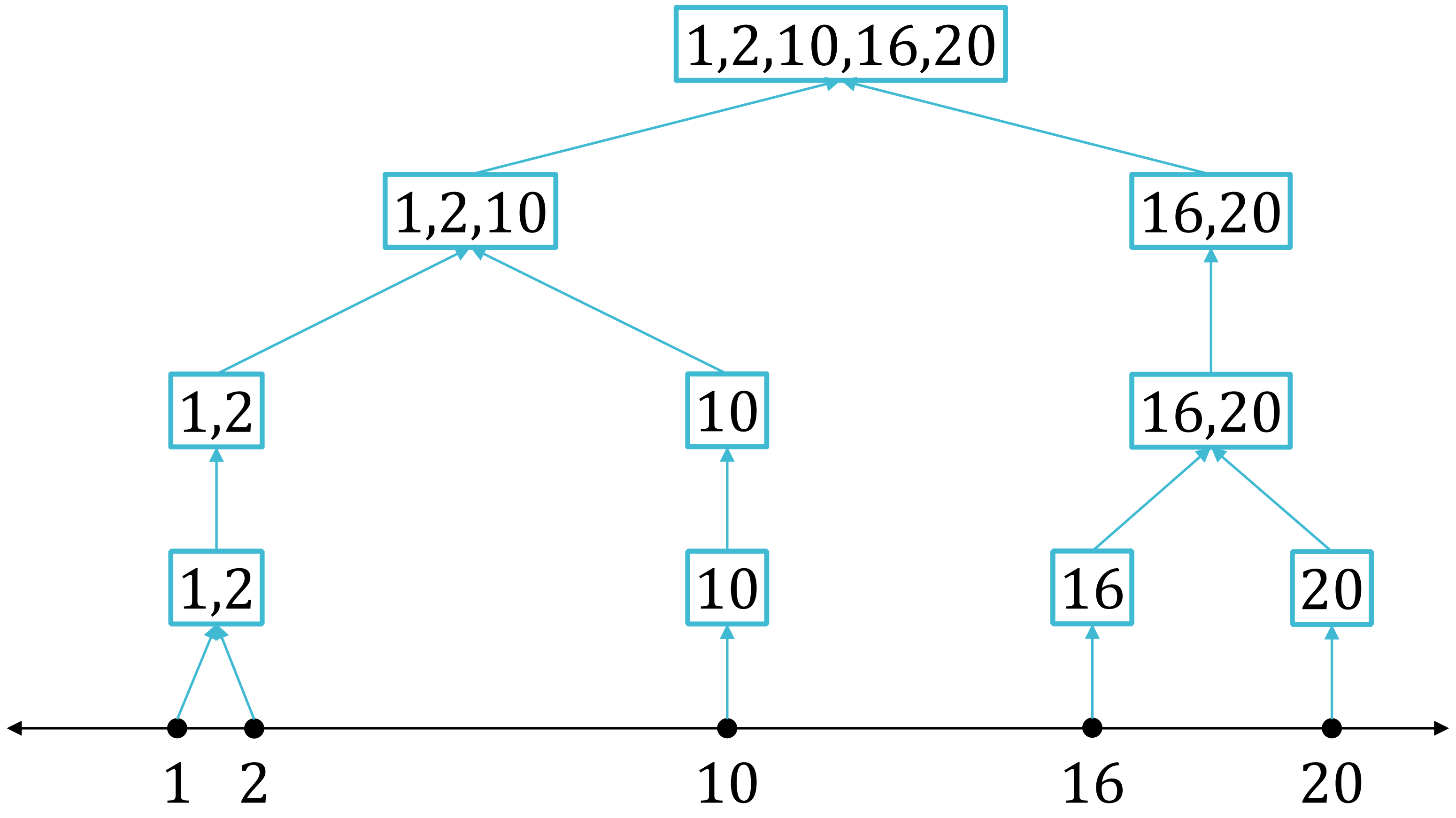
  - Complete-linkage: consider the farthest data points

$$d_{CL}\left(C_i, C_j\right) = \max_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} d(\boldsymbol{x}, \boldsymbol{y})$$

Single-Linkage Dendrogram



Complete-Linkage Dendrogram



Top-down Hierarchical Clustering

- Top-down (divisive)
  - Start with all data points in one cluster
  - Iteratively ***split*** the largest cluster into two clusters
  - Stop when all clusters are single data points
- Key question: how can we partition a cluster?

Recipe for K-means

- Define a model and model parameters
  - Assume K clusters and use the Euclidean distance
  - Parameters: $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ and $z^{(1)}, \dots, z^{(N)}$

- Write down an objective function

$$\sum_{n=1}^{N} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{\mu}_{z^{(n)}} \right\|_2$$

- Optimize the objective w.r.t. the model parameters
  - Use (block) coordinate descent

Block Coordinate Descent

- Goal: minimize some objective

$$\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}} = \operatorname{argmin} \ J(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

- Idea: iteratively pick one *block* of variables ($\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$) and minimize the objective w.r.t. that block, keeping the other(s) fixed.
  - Ideally, blocks should be the largest possible set of variables that can be efficiently optimized simultaneously

K-means Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}\right)\right\}_{n=1}^{N}, K$

1. Initialize cluster centers $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$
2. While NOT CONVERGED
   a. Assign each data point to the cluster with the nearest cluster center:
   $$z^{(n)} = \underset{k}{\operatorname{argmin}} \left\|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\right\|_2$$
   b. Recompute the cluster centers:
   $$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n\,:z^{(n)}=k} \boldsymbol{x}^{(n)}$$
   where N_k is the number of data points in cluster k

- Output: cluster centers $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ and cluster assignments $z^{(1)}, \dots, z^{(N)}$

Practice Problem: K-means

Given the initial cluster centers shown below, circle the image that depicts the cluster center positions after 1 iteration of the Lloyd's method

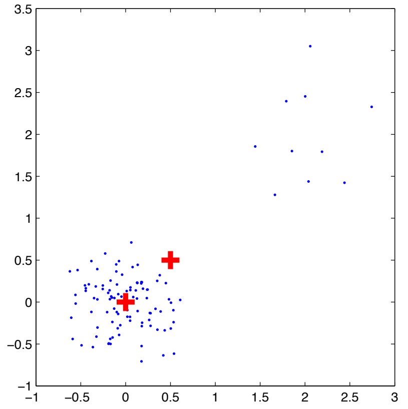
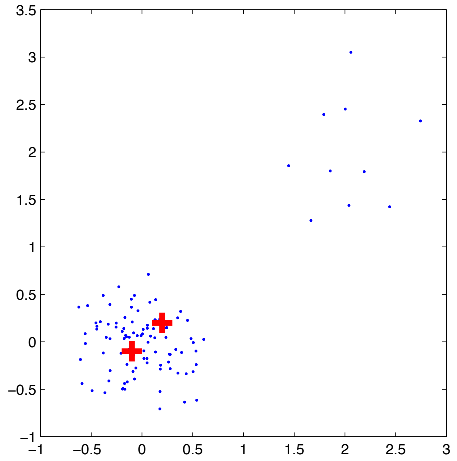
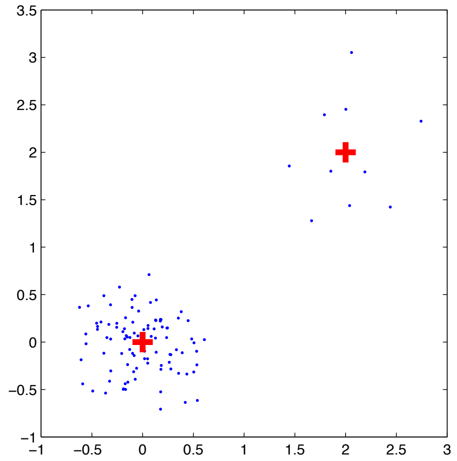
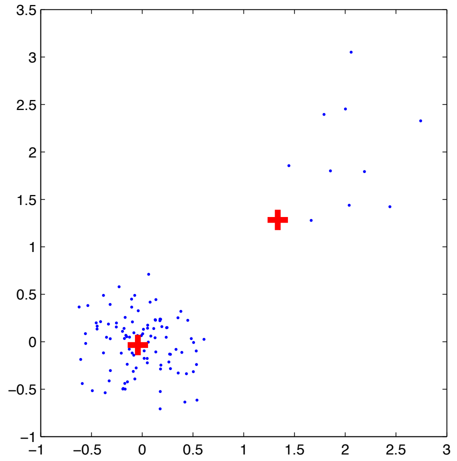
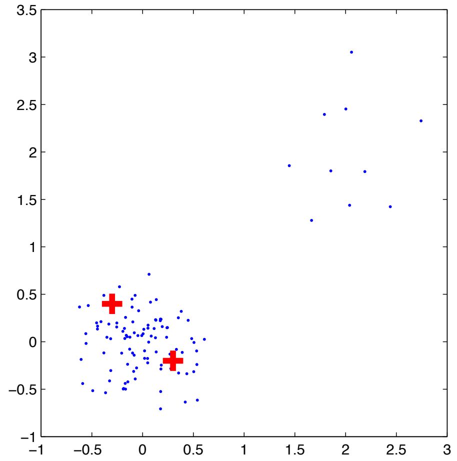


Figure 2: Initial data and cluster centers



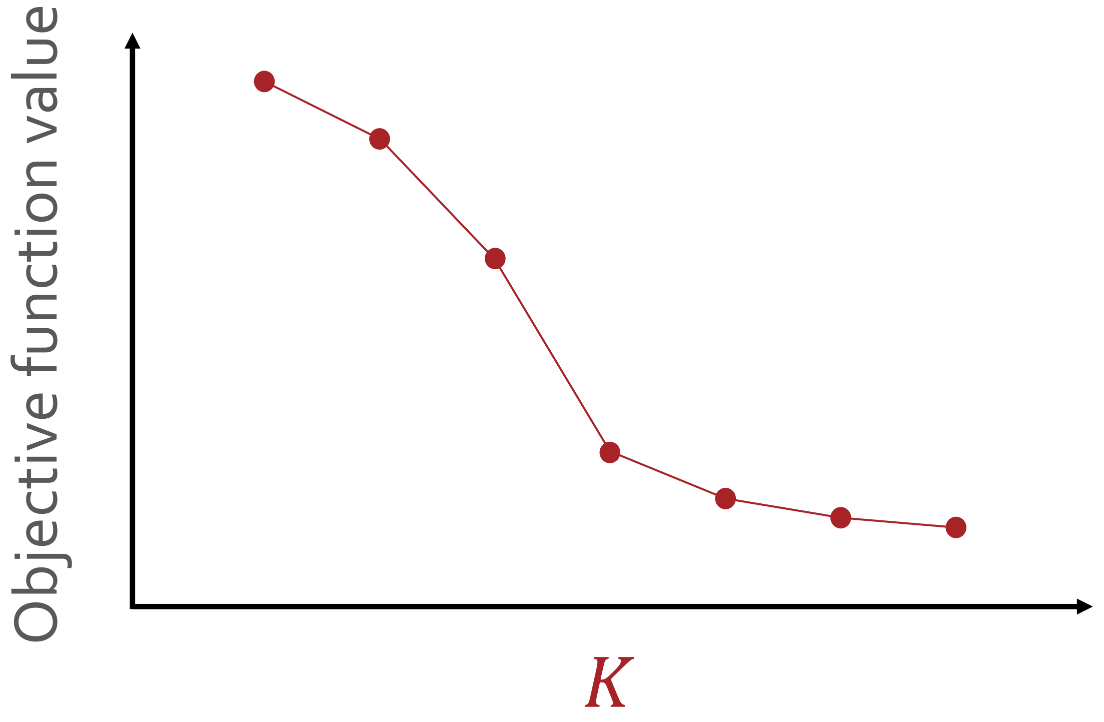






Setting K

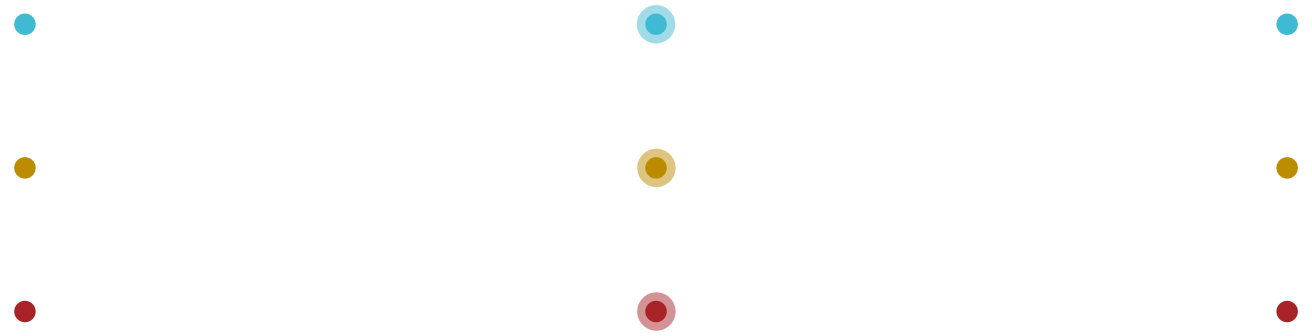
- Idea: choose the value of K that minimizes the objective function



- Look for the characteristic "elbow" or largest decrease when going from $K-1$ to K

Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



- Lloyd's method converges to a local minimum and that local minimum can be arbitrarily bad (relative to the optimal clusters)
- Intuition: want initial cluster centers to be far apart from one another

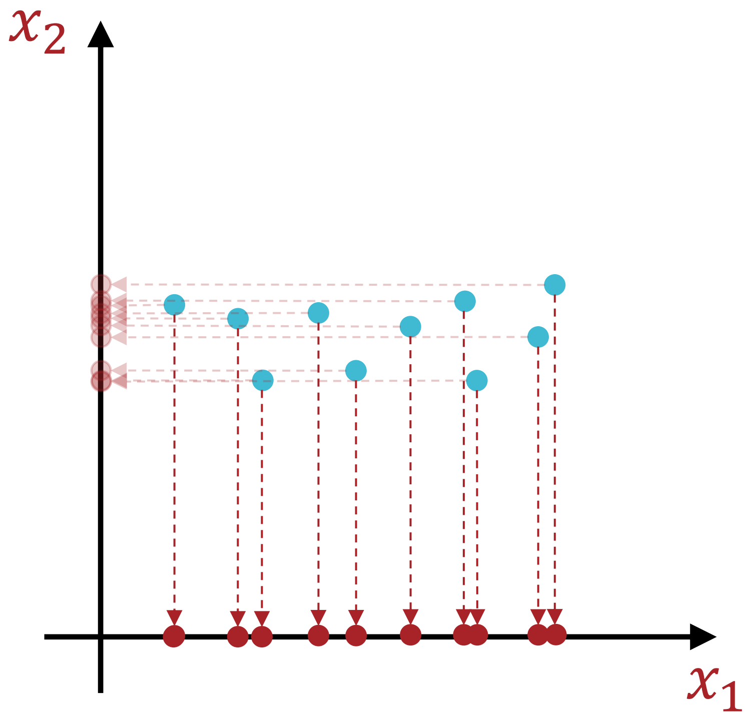
K-means++ (Arthur and Vassilvitskii, 2007)

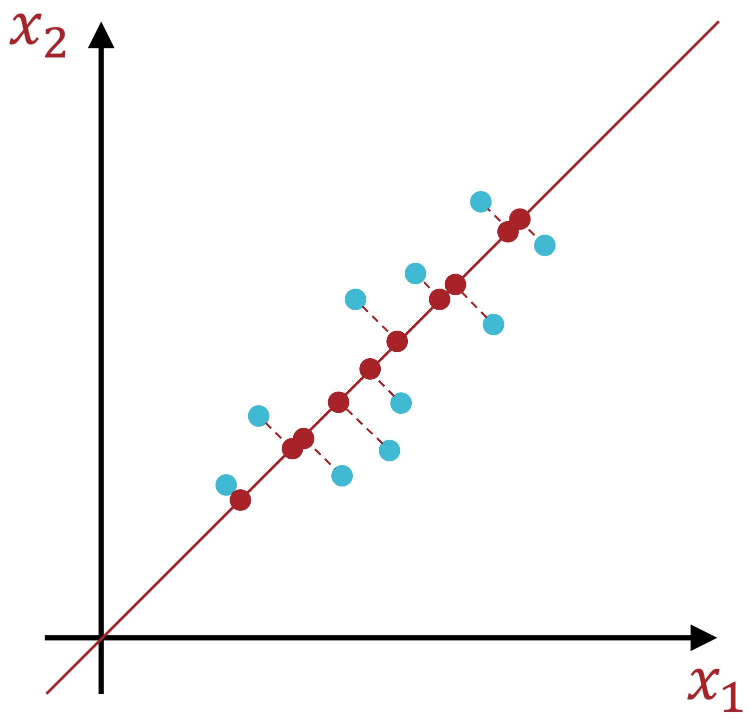
1. Choose the first cluster center randomly from the data points.
2. For each other data point $\boldsymbol{x}$, compute $D(\boldsymbol{x})$, the distance between $\boldsymbol{x}$ and the closest cluster center.
3. Select the next cluster center proportional to $D(\boldsymbol{x})^2$.
4. Repeat 2 and 3 $K - 1$ times.

- K-means++ achieves a $O(\log K)$ approximation to the optimal clustering in expectation
- Both Lloyd's method and K-means++ can benefit from multiple random restarts.

Dimensionality Reduction

- Goal: given some unlabeled data set, learn a latent (typically lower-dimensional) representation
- Use cases:
  - Reducing computational cost (runtime, storage, etc…)
  - Improving generalization
  - Visualizing data
- Applications:
  - High-resolution images/videos
  - Text data
  - Financial or transaction data





Feature Elimination ∈ Dimensionality Reduction

Centering the Data

- To be consistent, we will constrain principal components to be *orthogonal unit vectors* that begin at the origin
- Preprocess data to be centered around the origin:

1. $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}^{(n)}$
2. $\widetilde{\boldsymbol{x}}^{(n)} = \boldsymbol{x}^{(n)} - \boldsymbol{\mu} \ \forall\, n$
3. $X = \begin{bmatrix} \widetilde{\boldsymbol{x}}^{(1)^T} \\ \widetilde{\boldsymbol{x}}^{(2)^T} \\ \vdots \\ \widetilde{\boldsymbol{x}}^{(N)^T} \end{bmatrix}$

Minimizing the Reconstruction Error $\Updownarrow$ Maximizing the Variance

$$\hat{\boldsymbol{v}} = \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmin}} \sum_{n=1}^{N} \left\| \tilde{\boldsymbol{x}}^{(n)} - \left(\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}\right)\boldsymbol{v} \right\|_2^2$$

$$= \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmin}} \sum_{n=1}^{N} \left\| \tilde{\boldsymbol{x}}^{(n)} \right\|_2^2 - \left(\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}\right)^2$$

$$= \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmax}} \sum_{n=1}^{N} \left(\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}\right)^2 \longleftarrow$$

Variance of projections ($\tilde{\boldsymbol{x}}^{(n)}$ are centered)

$$= \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmax}} \boldsymbol{v}^T \left(\sum_{n=1}^{N} \tilde{\boldsymbol{x}}^{(n)} \tilde{\boldsymbol{x}}^{(n)^T} \right) \boldsymbol{v}$$

$$= \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmax}} \boldsymbol{v}^T (X^T X) \boldsymbol{v}$$

Maximizing the Variance

$$\hat{\boldsymbol{v}} = \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmax}} \boldsymbol{v}^T(X^TX)\boldsymbol{v}$$

$$(X^TX)\hat{\boldsymbol{v}} = \lambda\hat{\boldsymbol{v}} \rightarrow \hat{\boldsymbol{v}}^T(X^TX)\hat{\boldsymbol{v}} = \lambda\hat{\boldsymbol{v}}^T\hat{\boldsymbol{v}} = \lambda$$

- The first principal component is the eigenvector $\hat{\boldsymbol{v}}_1$ that corresponds to the largest eigenvalue λ_1
- The second principal component is the eigenvector $\hat{\boldsymbol{v}}_2$ that corresponds to the second largest eigenvalue λ_1
  - $\hat{\boldsymbol{v}}_1$ and $\hat{\boldsymbol{v}}_2$ are orthogonal
- Etc …
- λ_i is a measure of how much variance falls along $\hat{\boldsymbol{v}}_i$

PCA Algorithm

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)})\}_{n=1}^{N}, \rho$

1. Center the data
2. Use SVD to compute the eigenvalues and eigenvectors of $X^T X$
3. Collect the top ρ eigenvectors (corresponding to the ρ largest eigenvalues), $V_\rho \in \mathbb{R}^{D \times \rho}$
4. Project the data into the space defined by V_ρ, $Z = XV_\rho$

- Output: Z, the transformed (potentially lower-dimensional) data

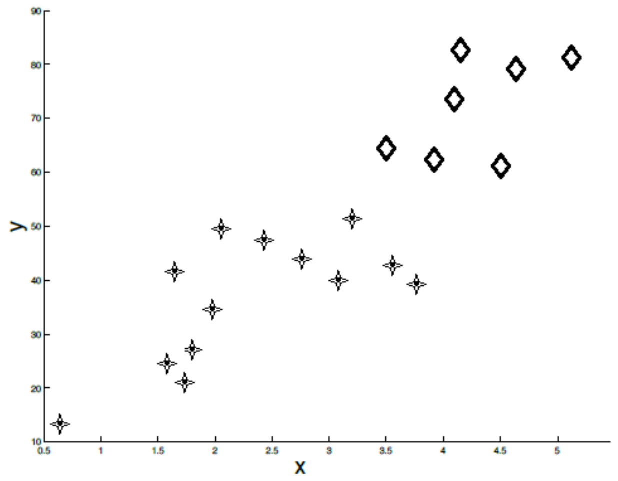
Practice Problem: PCA

In the following plots, a train set of data points X belonging to two classes on $\mathbb{R}^2$ are given, where the original features are the coordinates (x, y). For each, answer the following questions:

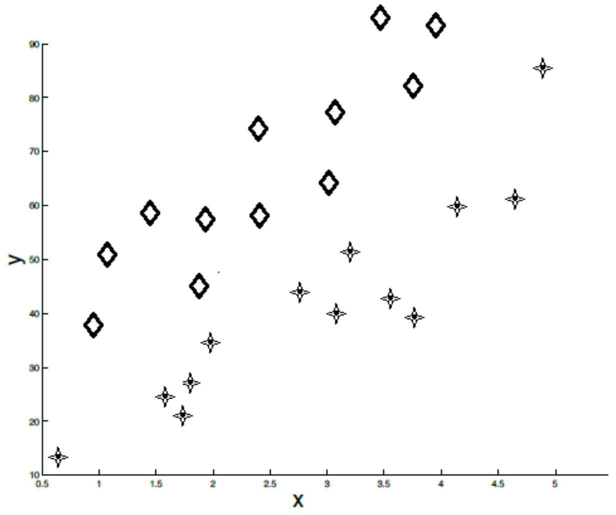
Draw all the principal components.

Can we correctly classify this dataset by using a threshold function after projecting onto one of the principal components? If so, which principal component should we project onto? If not, explain in 1–2 sentences why it is not possible.

Dataset 1:



Dataset 2:



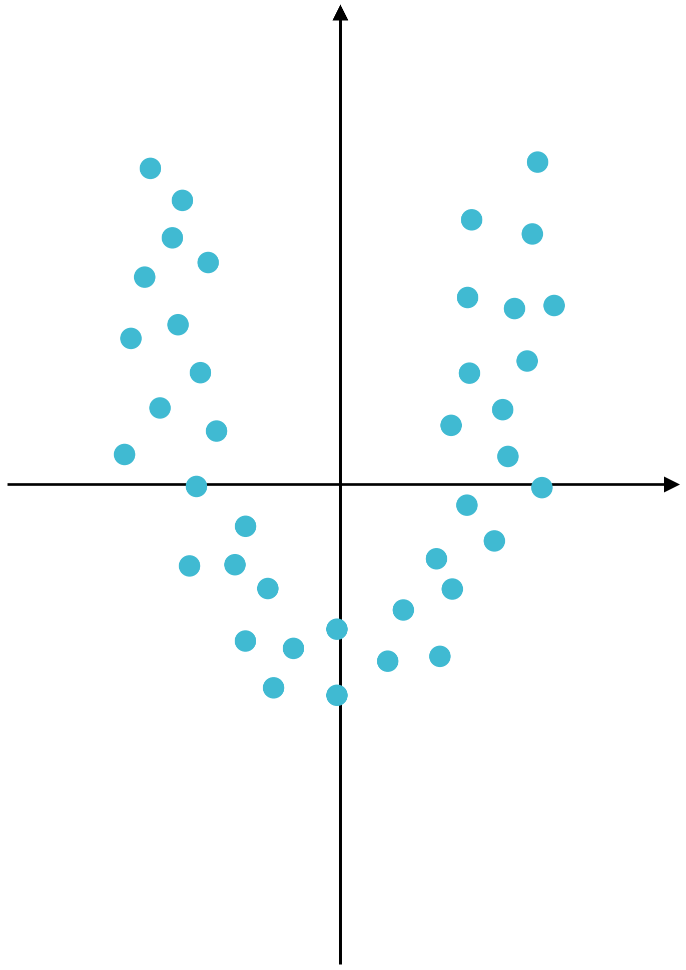
Choosing the number of PCs

- Define a percentage of explained variance for the i^{th} PC:

$$\lambda_i \big/ \sum \lambda_j$$

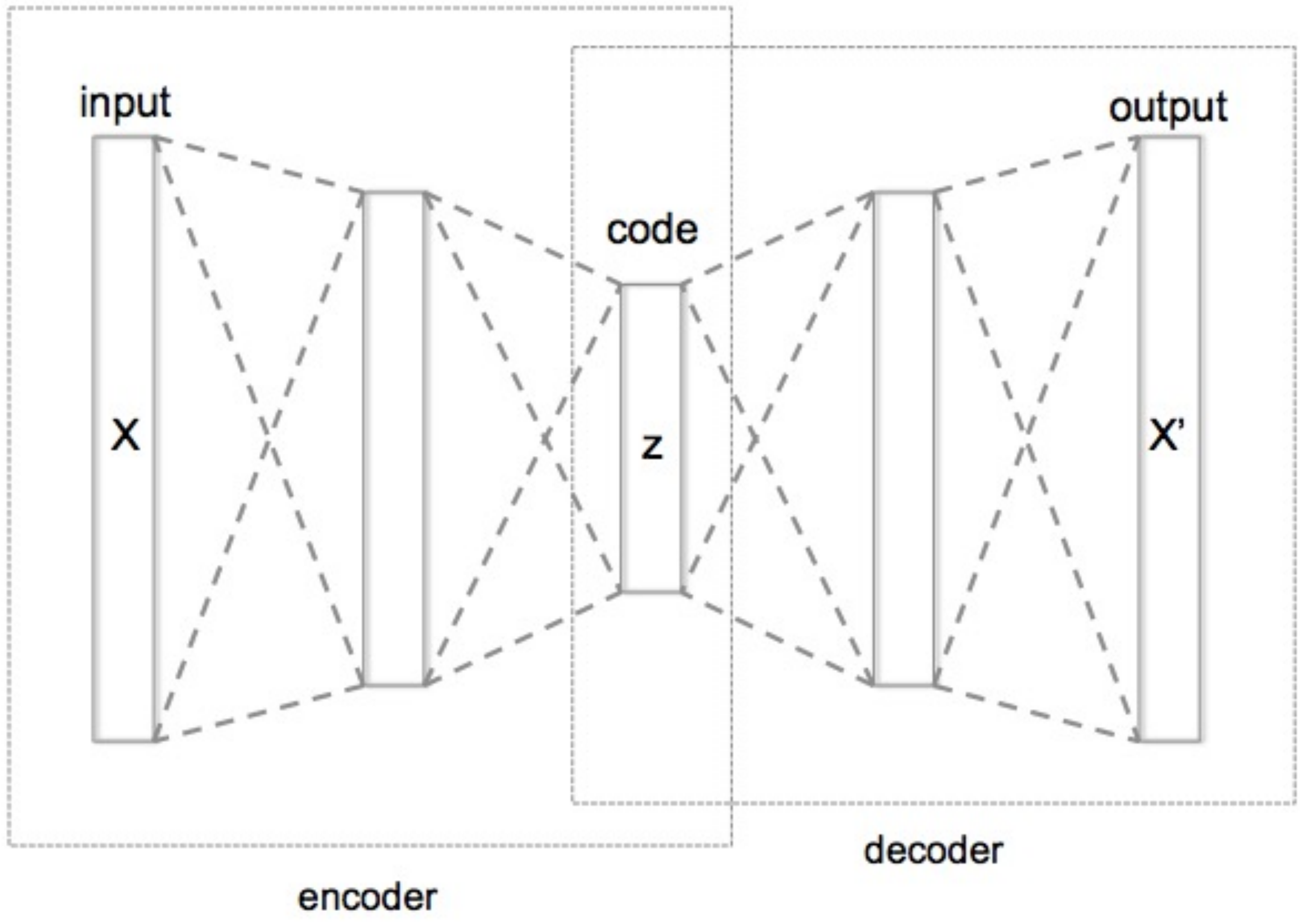
- Select all PCs above some threshold of explained variance, e.g., 5%
- Keep selecting PCs until the total explained variance exceeds some threshold, e.g., 90%
- Evaluate on some downstream metric

Shortcomings of PCA



- Principal components are orthogonal (unit) vectors
- Principal components can be expressed as linear combinations of the data

Deep Autoencoders



Source: https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png

Decision Trees: Pros & Cons

- Pros
  - Interpretable
  - Efficient (computational cost and storage)
  - Can be used for classification and regression tasks
  - Compatible with categorical and real-valued features
- Cons
  - Learned greedily: each split only considers the immediate impact on the splitting criterion
    - Not guaranteed to find the smallest (fewest number of splits) tree that achieves a training error rate of 0.
  - Prone to overfit
  - High variance
    - Can be addressed via ensembles $\rightarrow$ random forests

Random Forests

- Combines the prediction of many diverse decision trees to reduce their variability
- If B independent random variables $x^{(1)}, x^{(2)}, \dots, x^{(B)}$ all have variance σ^2, then the variance of $\frac{1}{B}\sum_{b=1}^{B} x^{(b)}$ is $\frac{\sigma^2}{B}$
- Random forests = bagging + split-feature randomization

  = <u>**b**</u>ootstrap <u>**agg**</u>regat<u>**ing**</u> + split-feature randomization

Aggregating

- How can we combine multiple decision trees, $\{t_1, t_2, \dots, t_B\}$, to arrive at a single prediction?
- Regression - average the predictions:

$$\bar{t}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} t_b(\boldsymbol{x})$$

- Classification - plurality (or majority) vote; for binary labels encoded as $\{-1, +1\}$:

$$\bar{t}(\boldsymbol{x}) = \operatorname{sign}\left(\frac{1}{B}\sum_{b=1}^{B} t_b(\boldsymbol{x})\right)$$

Bootstrapping

- Idea: resample the data multiple times ***with replacement***
  - Each bootstrapped sample has the same number of data points as the original data set
  - Duplicated points cause different decision trees to focus on different parts of the input space

MovieID	⋯
1	⋯
2	⋯
3	⋯
⋮	⋮
19	⋯
20	⋯

Training data

MovieID	⋯
1	⋯
1	⋯
1	⋯
⋮	⋮
14	⋯
19	⋯

Bootstrapped Sample 1

MovieID	⋯
4	⋯
4	⋯
5	⋯
⋮	⋮
16	⋯
16	⋯

Bootstrapped Sample 2

⋯

⋯

Split-feature Randomization

- Issue: decision trees trained on bootstrapped samples still behave similarly
- Idea: in addition to sampling the data points (i.e., the rows), also sample the features (i.e., the columns)
- Each time a split is being considered, limit the possible features to a randomly sampled subset

Runtime	Genre	Budget	Year	IMDB	Rating

Random Forests

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, B, \rho$
- For $b = 1, 2, \ldots, B$
  - Create a dataset, $\mathcal{D}_b$, by sampling N points from the original training data $\mathcal{D}$ ***with replacement***
  - Learn a decision tree, t_b, using $\mathcal{D}_b$ and the ID3 algorithm ***with split-feature randomization***, sampling ρ features for each split
- Output: $\bar{t} = f(t_1, \ldots, t_B)$, the aggregated hypothesis

Practice Problem: Random Forests

- Suppose you fix ρ, the number of features used for split-feature randomization, and increase B, the number of trees in the random forest: will the variance of the random forest tend to increase, decrease or stay the same? Briefly justify your answer in 2-3 concise sentences.

Out-of-bag Error

- For each training point, $\boldsymbol{x}^{(n)}$, there are some decision trees which $\boldsymbol{x}^{(n)}$ was not used to train (roughly B/e trees or 37%)
  - Let these be $t^{(-n)} = \left\{t_1^{(-n)}, t_2^{(-n)}, \dots, t_{N_{-n}}^{(-n)}\right\}$
- Compute an aggregated prediction for each $\boldsymbol{x}^{(n)}$ using the trees in $t^{(-n)}$, $\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right)$
- Compute the out-of-bag (OOB) error, e.g., for classification

$$E_{OOB} = \frac{1}{N}\sum_{n=1}^{N}\left[\!\left[\bar{t}^{(-n)}\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)}\right]\!\right]$$

- E_{OOB} can be used for hyperparameter optimization!

Feature Importance

- Some of the interpretability of decision trees gets lost when switching to random forests
- Random forests allow for the computation of “feature importance”, a way of ranking features based on how useful they are at predicting the target
- Initialize each feature’s importance to zero
- Each time a feature is chosen to be split on, add the reduction in Gini impurity (weighted by the number of data points in the split) to its importance

Decision Trees: Pros & Cons

- Pros
    - Interpretable
    - Efficient (computational cost and storage)
    - Can be used for classification and regression tasks
    - Compatible with categorical and real-valued features
- Cons
    - Learned greedily: each split only considers the immediate impact on the splitting criterion
        - Not guaranteed to find the smallest (fewest number of splits) tree that achieves a training error rate of 0.
    - Prone to overfit
    - High variance
        - Can be addressed via bagging $\rightarrow$ random forests
    - High bias (especially short trees, i.e., stumps)
        - Can be addressed via boosting

Boosting

- Another ensemble method (like bagging) that combines the predictions of multiple hypotheses.
- Aims to reduce the bias of a “weak” or highly biased model (can also reduce variance).

AdaBoost

- Input: $\mathcal{D}\ \left(y^{(n)} \in \{-1,+1\}\right)$, T
- Initialize data point weights: $\omega_0^{(1)}, \ldots, \omega_0^{(N)} = \frac{1}{N}$
- For $t = 1, \ldots, T$
  1. Train a weak learner, h_t, by minimizing the *weighted* training error
  2. Compute the *weighted* training error of h_t:
$$\epsilon_t = \sum_{n=1}^{N} \omega_{t-1}^{(n)} \mathbb{1}\left(y^{(n)} \neq h_t\left(\boldsymbol{x}^{(n)}\right)\right)$$
  3. Compute the **importance** of h_t:
$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$
  4. Update the data point weights:
$$\omega_t^{(n)} = \frac{\omega_{t-1}^{(n)}}{Z_t} \times \begin{cases} e^{-\alpha_t} \text{ if } h_t\left(\boldsymbol{x}^{(n)}\right) = y^{(n)} \\ e^{\alpha_t} \quad \text{if } h_t\left(\boldsymbol{x}^{(n)}\right) \neq y^{(n)} \end{cases} = \frac{\omega_{t-1}^{(n)} e^{-\alpha_t y^{(n)} h_t\left(\boldsymbol{x}^{(n)}\right)}}{Z_t}$$

- Output: an aggregated hypothesis

$$g_T(\boldsymbol{x}) = \text{sign}\left(H_T(\boldsymbol{x})\right)$$
$$= \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$$

Why AdaBoost?

1. If you want to use weak learners …
2. … and want your final hypothesis to be a weighted combination of weak learners, …
3. … then Adaboost greedily minimizes the exponential loss:

$$e(h(\boldsymbol{x}), y) = e^{(-yh(\boldsymbol{x}))}$$

1. Because they're low variance / computational constraints
2. Because weak learners are not great on their own
3. Because the exponential loss upper bounds binary error

Exponential Loss

- Claim:

$$\frac{1}{N}\sum_{n=1}^{N} e^{\left(-y^{(n)} h\left(\boldsymbol{x}^{(n)}\right)\right)} \geq \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\left(\text{sign}\left(h\left(\boldsymbol{x}^{(n)}\right)\right) \neq y^{(n)}\right)$$

- Consequence:

$$\frac{1}{N}\sum_{n=1}^{N} e^{\left(-y^{(n)} h\left(\boldsymbol{x}^{(n)}\right)\right)} \to 0$$

$$\Longrightarrow \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\left(\text{sign}\left(h\left(\boldsymbol{x}^{(n)}\right)\right) \neq y^{(n)}\right) \to 0$$

Exponential Loss

- Claim: if $g_T = \text{sign}(H_T)$ is the Adaboost hypothesis, then

$$\frac{1}{N}\sum_{n=1}^{N} e^{\left(-y^{(n)}H_T\left(\boldsymbol{x}^{(n)}\right)\right)} = \prod_{t=1}^{T} Z_t$$

- Proof:

$$\omega_0^{(n)} = \frac{1}{N},\ \omega_1^{(n)} = \frac{e^{-\alpha_1 y^{(n)} h_1\left(\boldsymbol{x}^{(n)}\right)}}{NZ_1},\ \omega_2^{(n)} = \frac{e^{-\alpha_1 y^{(n)} h_1\left(\boldsymbol{x}^{(n)}\right)} e^{-\alpha_2 y^{(n)} h_2\left(\boldsymbol{x}^{(n)}\right)}}{NZ_1Z_2}$$

$$\omega_T^{(n)} = \frac{\prod_{t=1}^{T} e^{-\alpha_t y^{(n)} h_t\left(\boldsymbol{x}^{(n)}\right)}}{N\prod_{t=1}^{T} Z_t} = \frac{e^{-y^{(n)}\sum_{t=1}^{T}\alpha_t h_t\left(\boldsymbol{x}^{(n)}\right)}}{N\prod_{t=1}^{T} Z_t} = \frac{e^{-y^{(n)}H_T\left(\boldsymbol{x}^{(n)}\right)}}{N\prod_{t=1}^{T} Z_t}$$

$$\sum_{n=1}^{N}\omega_T^{(n)} = \sum_{n=1}^{N}\frac{e^{-y^{(n)}H_T\left(\boldsymbol{x}^{(n)}\right)}}{N\prod_{t=1}^{T} Z_t} = 1 \Longrightarrow \frac{1}{N}\sum_{n=1}^{N} e^{-y^{(n)}H_T\left(\boldsymbol{x}^{(n)}\right)} = \prod_{t=1}^{T} Z_t \ \blacksquare$$

Exponential Loss

- Claim: if $g_T = \text{sign}(H_T)$ is the Adaboost hypothesis, then

$$\frac{1}{N}\sum_{n=1}^{N} e^{\left(-y^{(n)}H_T\left(\boldsymbol{x}^{(n)}\right)\right)} = \prod_{t=1}^{T} Z_t$$

- Consequence: one way to minimize the exponential training loss is to greedily minimize Z_t, i.e., in each iteration, make the normalization constant as small as possible by tuning α_t.

Greedy Exponential Loss Minimization

$$Z_t = e^{-a}(1 - \epsilon_t) + e^{a}\epsilon_t$$

$$\frac{\partial Z_t}{\partial a} = -e^{-a}(1 - \epsilon_t) + e^{a}\epsilon_t \Longrightarrow -e^{-\hat{a}}(1 - \epsilon_t) + e^{\hat{a}}\epsilon_t = 0$$

$$\Longrightarrow e^{\hat{a}}\epsilon_t = e^{-\hat{a}}(1 - \epsilon_t)$$

$$\Longrightarrow e^{2\hat{a}} = \frac{1 - \epsilon_t}{\epsilon_t}$$

$$\Longrightarrow \hat{a} = \frac{1}{2}\log\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) = \alpha_t$$

$$\frac{\partial^2 Z_t}{\partial a^2} = e^{-a}(1 - \epsilon_t) + e^{a}\epsilon_t > 0$$

Training Error

$$\frac{1}{N}\sum_{n=1}^{N}\mathbb{1}\left(y^{(n)} \neq g_T\left(\boldsymbol{x}^{(n)}\right)\right) \leq \frac{1}{N}\sum_{n=1}^{N} e^{\left(-y^{(n)}H_T\left(\boldsymbol{x}^{(n)}\right)\right)}$$

$$= \prod_{t=1}^{T} Z_t$$

$$= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \rightarrow 0 \text{ as T} \rightarrow \infty$$

$$\left(\text{as long as } \epsilon_t < \frac{1}{2}\ \forall\, t\right)$$