CS 189 Final Review

Fall 2024

Pre-midterm topics

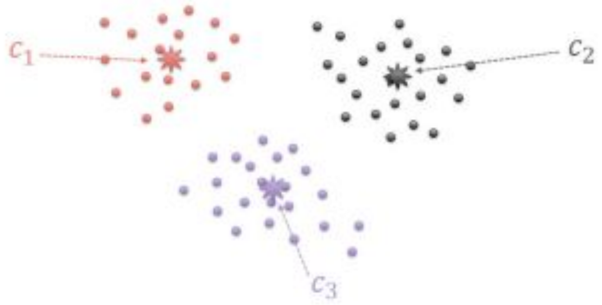
Review Slides

Clustering

K-means clustering

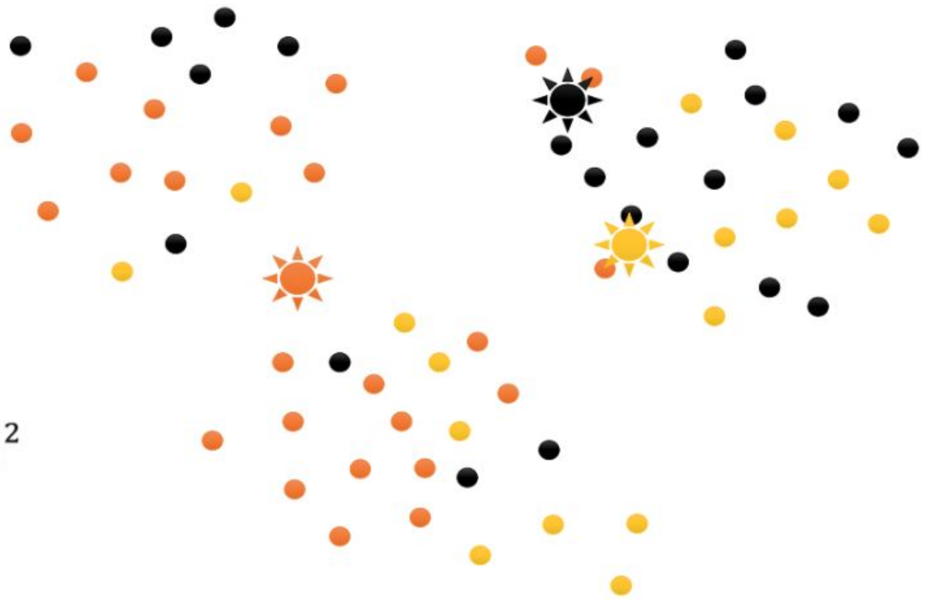
- Assign points to clusters by minimizing distance to centroids

$$\underset{\underbrace{S=C_1\cup\ldots\cup C_K}_{\text{cluster partition}},\ \underbrace{\{c_1,\ldots,c_K\}}_{\text{cluster centroids}}}{\operatorname{argmin}} \sum_k \sum_{x\in C_k} \left\|x-c_k\right\|^2$$



1. Compute partition by choosing closest centroid
2. Compute centers by averaging over partition
3. Continue until centers do not change

K-means example

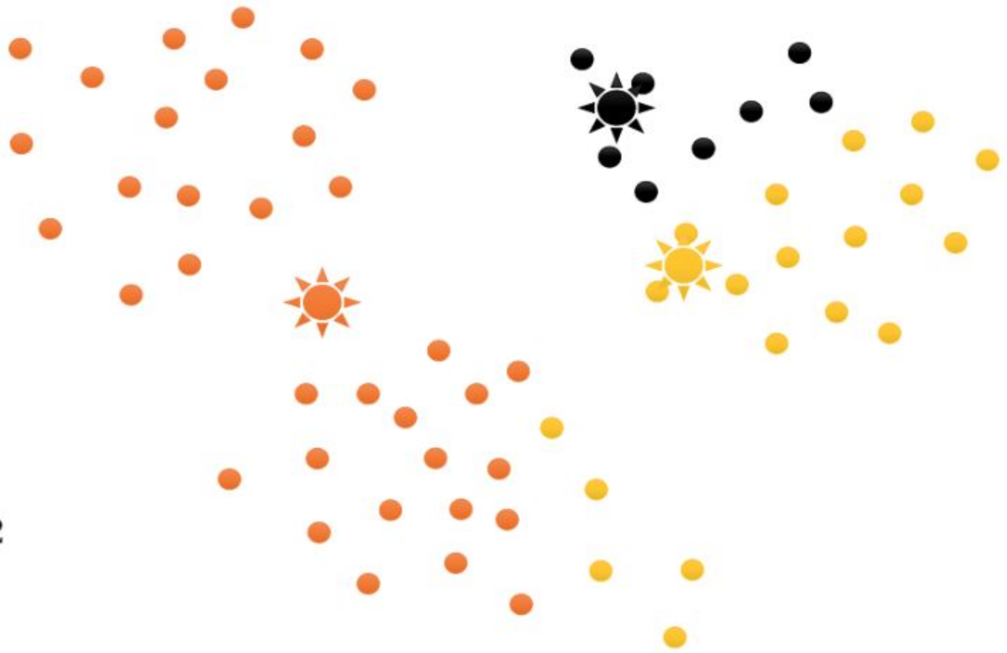


$$\hat{c}_k = \underset{c_k}{\operatorname{argmin}} \sum_{x \in C_k} \|x - c_k\|^2$$

$$\Rightarrow \hat{c}_k = \frac{1}{N} \sum_{x \in C_k} x$$

[slide courtesy Yisong Yue]

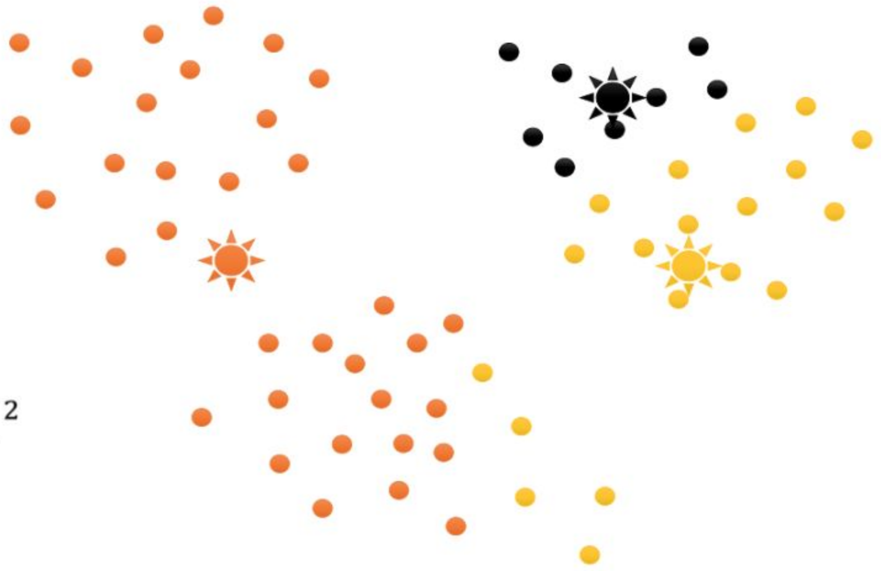
K-means example



$$z_i \equiv \underset{k}{\operatorname{argmin}} \|x_i - c_k\|^2$$

$$\Rightarrow \hat{C}_k = \{x_i | z_i = k\}.$$

K-means example

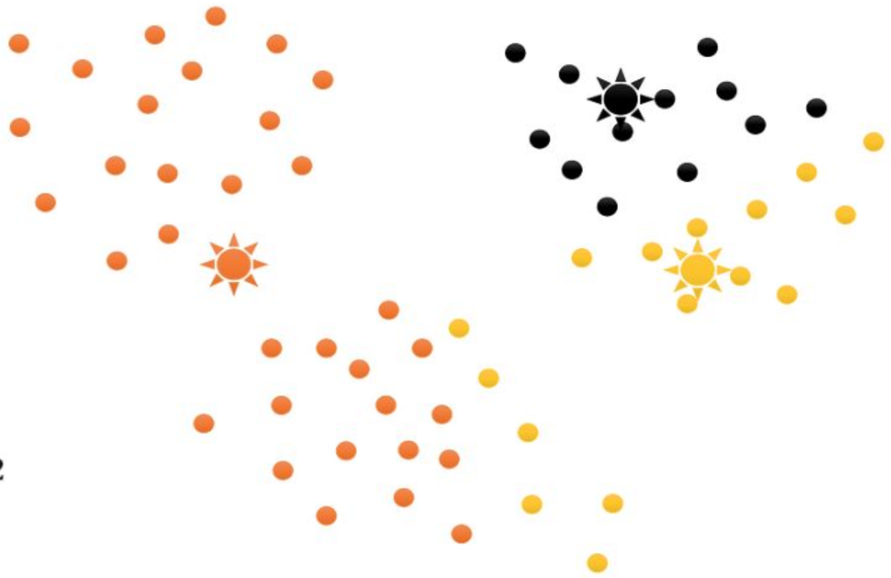


$$\hat{c}_k = \underset{c_k}{\operatorname{argmin}} \sum_{x \in C_k} \|x - c_k\|^2$$

$$\Rightarrow \hat{c}_k = \frac{1}{N} \sum_{x \in C_k} x$$

[slide courtesy Yisong Yue]

K-means example

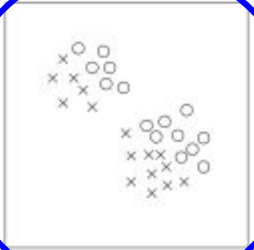


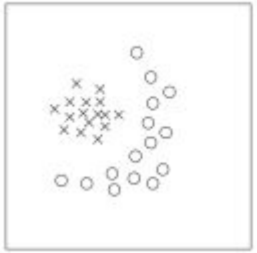
$$z_i \equiv \underset{k}{\operatorname{argmin}} \|x_i - c_k\|^2$$
$$\Rightarrow \hat{C}_k = \{x_i | z_i = k\}.$$

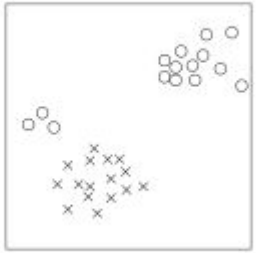
Practice Question

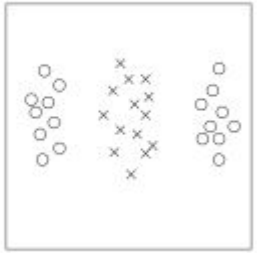
Fall 2023 Midterm, 1.11

11. Which of the following k-means cluster assignments could be a possible result after running k-means to convergence for 2 clusters?


[✓ (a)]

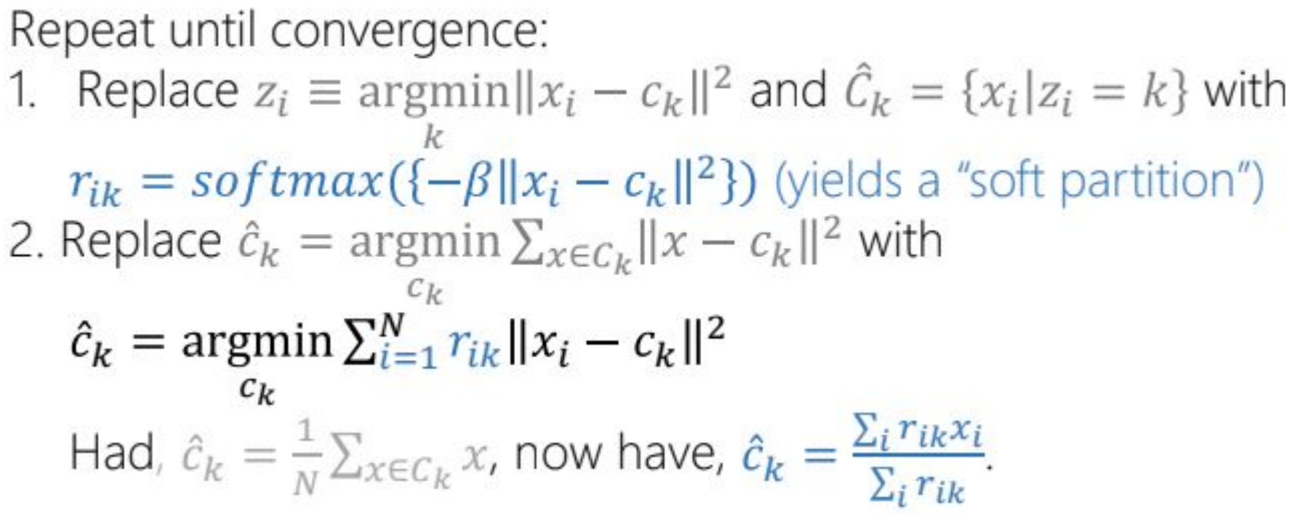

(b)


(c)


(d)

Soft k-means

- Probabilistic cluster assignment using softmax of distances



Repeat until convergence:

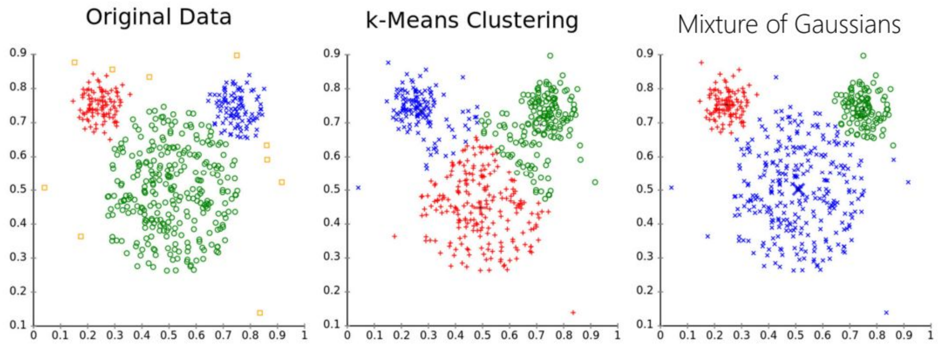
1. Replace $z_i \equiv \underset{k}{\operatorname{argmin}} \|x_i - c_k\|^2$ and $\hat{C}_k = \{x_i | z_i = k\}$ with
   $r_{ik} = softmax(\{-\beta \|x_i - c_k\|^2\})$ (yields a "soft partition")
2. Replace $\hat{c}_k = \underset{c_k}{\operatorname{argmin}} \sum_{x \in C_k} \|x - c_k\|^2$ with
   $$\hat{c}_k = \underset{c_k}{\operatorname{argmin}} \sum_{i=1}^{N} r_{ik} \|x_i - c_k\|^2$$
   Had, $\hat{c}_k = \frac{1}{N} \sum_{x \in C_k} x$, now have, $\hat{c}_k = \frac{\sum_i r_{ik} x_i}{\sum_i r_{ik}}$.

Mixture of Gaussians

- What if we probabilistically model each cluster as a (non-spherical) Gaussian
- Likelihood for each point is

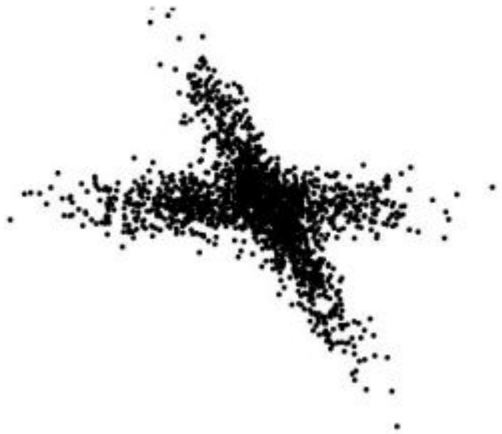
$$L_i = \sum_{k=1}^{K} P(x_i | x_i \in z_k) P(z_k) = \sum_{k=1}^{K} N(x_i | \mu_k, \Sigma_k) \alpha_k$$

- Learn the parameters $\mu_k, \Sigma_k, \alpha_k$



Practice Question

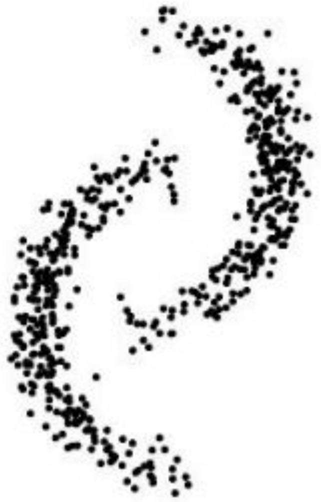
You want to cluster this 2D data into 2 clusters. Which of the these approaches, when used alone, would work well?



- [✓ Mixture of Gaussians]
- ○ K-means
- ○ Principle Components Analysis
- ○ Isomap
- ○ Class-conditional Gaussians

Practice Question

You want to cluster this 2D data into 2 clusters. Which of the these approaches, when used alone, is most likely to work well?



- ○ Mixture of Gaussians
- ○ K-means
- ○ PCA followed by Mixture of Gaussians
- [✓ Isomap followed by Mixture of Gaussians]
- ○ Class-conditional Gaussians
- ○ t-SNE

Kleinberg impossibility theorem

1. **Scale-invariance:** stretching the data should yield the same clustering
2. **Consistency:** stretching the space between clusters yields the same clustering
3. **Richness:** clustering should be able to produce any arbitrary partition

No clustering method can satisfy all three properties!

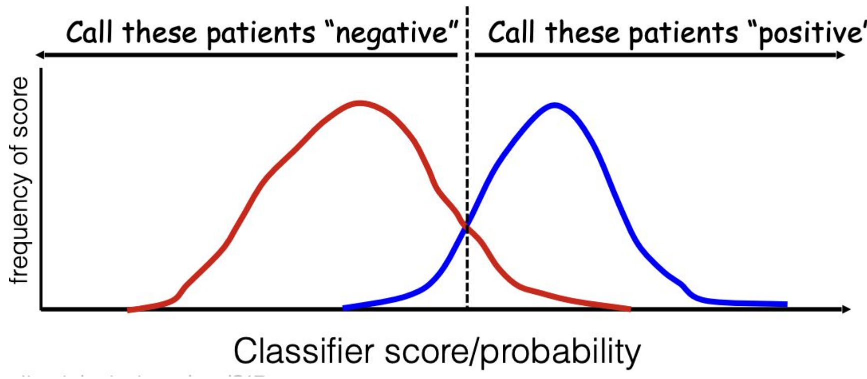
Model Evaluation

Classifier Decision Outcomes

- Possible binary classification results:
  - False positive (FP): predicted +1, truth -1
  - False negative (FN): predicted -1, truth +1
  - True positive (TP): predicted +1, truth +1
  - True negative (TN): predicted -1, truth -1

Distribution of classifier "scores" of healthy (-1) and unhealthy (1) individuals in a test set.

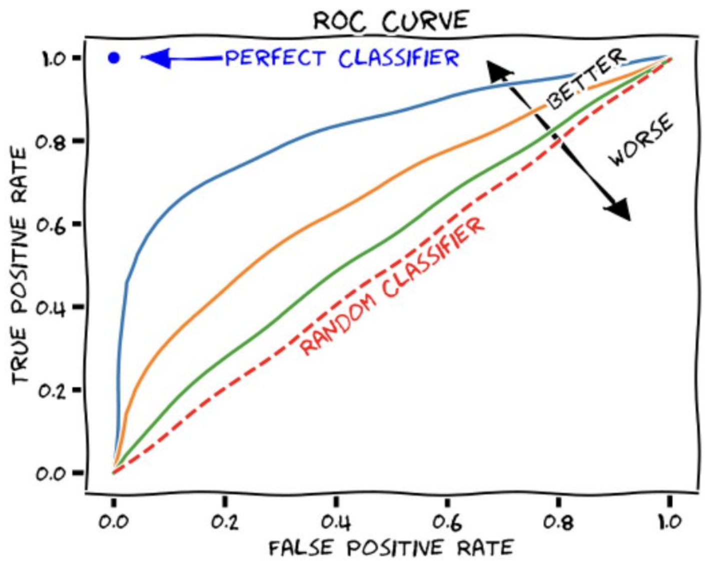
(score could be a probability, but need not be)



		MODEL PREDICTIONS	
		Negative	*Positive*
GROUND TRUTH	*Negative*	TN	FP
	Positive	FN	TP

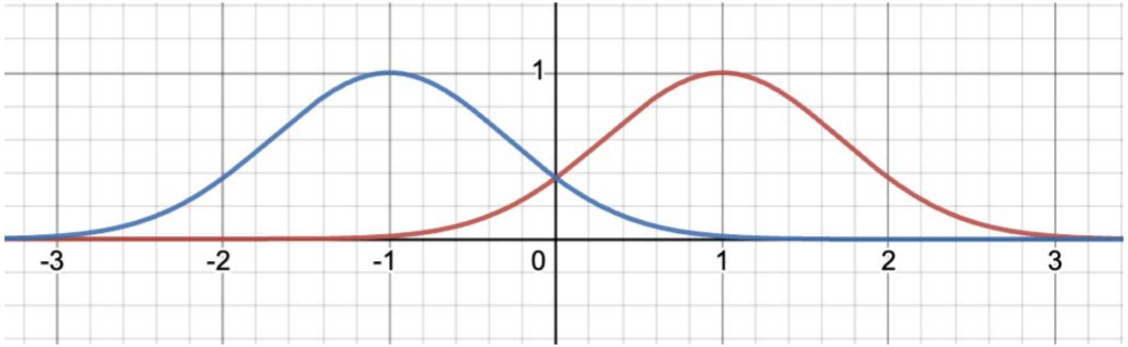
ROC Curves

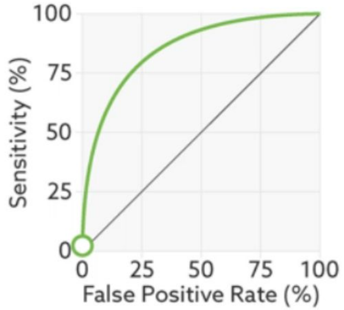
- Axes:
  - x-axis: FP rate (1-specificity)
  - y-axis: TP rate (sensitivity)
- Area under the curve (AUROC or AUC for short)
  - Larger area = better model
  - Probabilistic meaning?

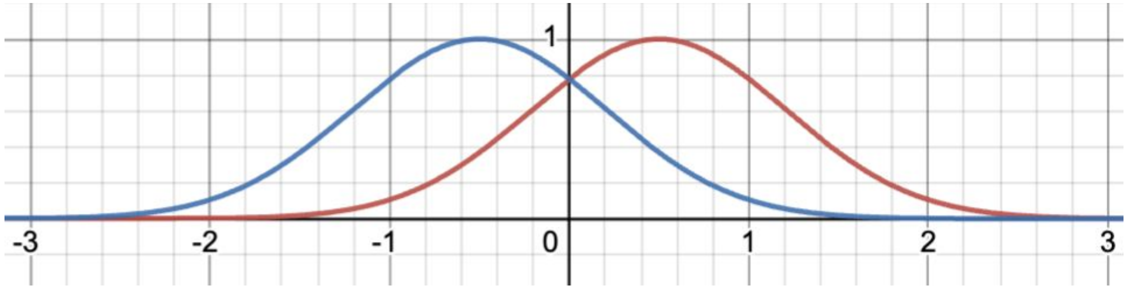


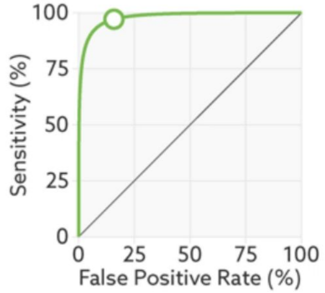
Practice Question

Match binary classifiers for each set of distributions to their ROC curves



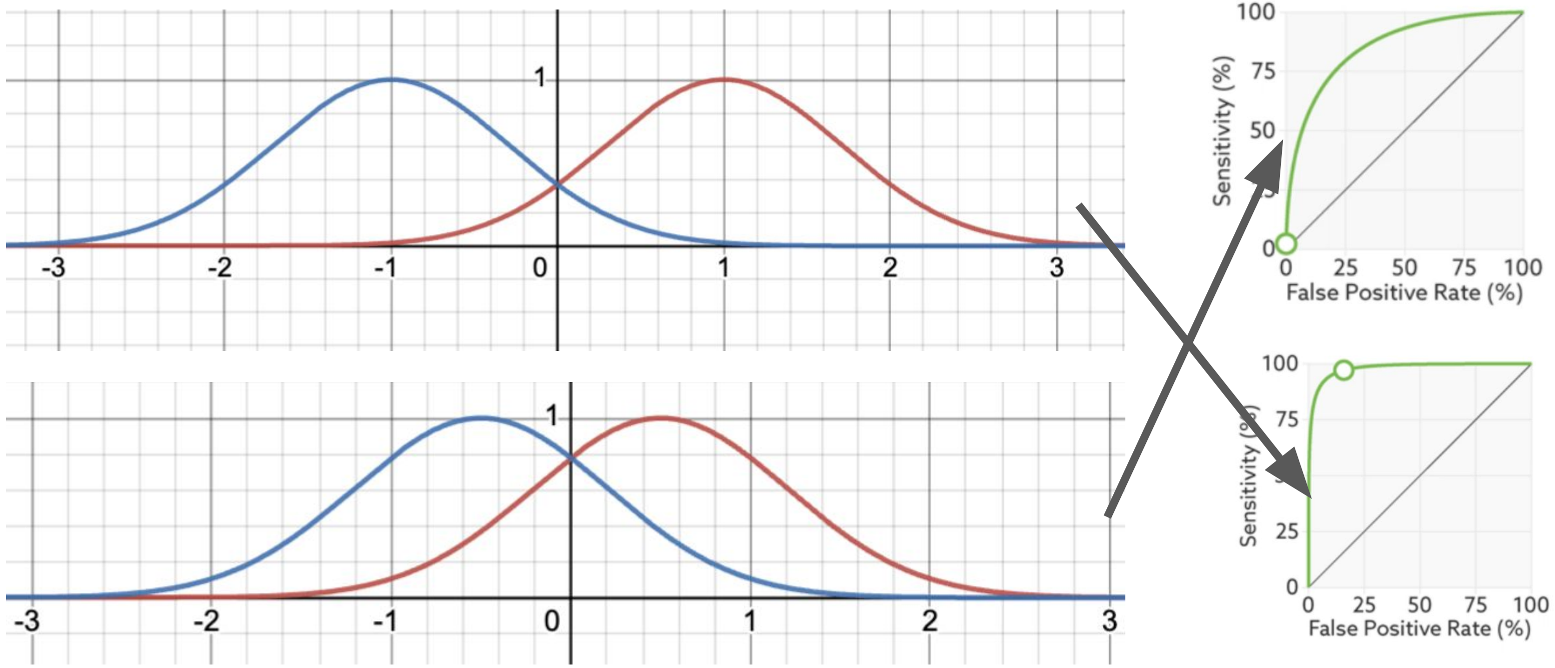






Solution

Further distributions allow for a better model

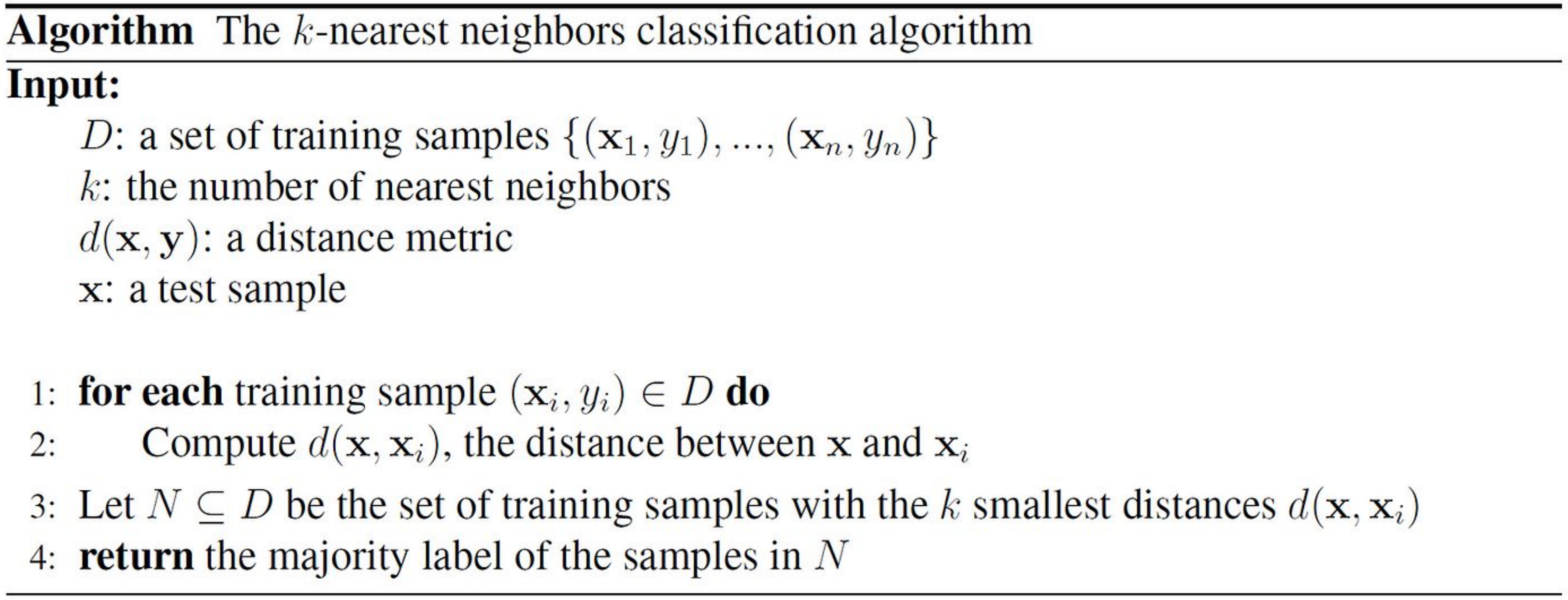


Nearest Neighbors

k-NN Algorithm

```
Algorithm The $k$-nearest neighbors classification algorithm
Input:
    $D$: a set of training samples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$
    $k$: the number of nearest neighbors
    $d(\mathbf{x}, \mathbf{y})$: a distance metric
    $\mathbf{x}$: a test sample
```

k-NN Algorithm



Algorithm The k-nearest neighbors classification algorithm

```
Input:
    $D$: a set of training samples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$
    $k$: the number of nearest neighbors
    $d(\mathbf{x}, \mathbf{y})$: a distance metric
    $\mathbf{x}$: a test sample

1: for each training sample $(\mathbf{x}_i, y_i) \in D$ do
2:     Compute $d(\mathbf{x}, \mathbf{x}_i)$, the distance between $\mathbf{x}$ and $\mathbf{x}_i$
3: Let $N \subseteq D$ be the set of training samples with the $k$ smallest distances $d(\mathbf{x}, \mathbf{x}_i)$
4: return the majority label of the samples in $N$
```

How do we choose *k?*

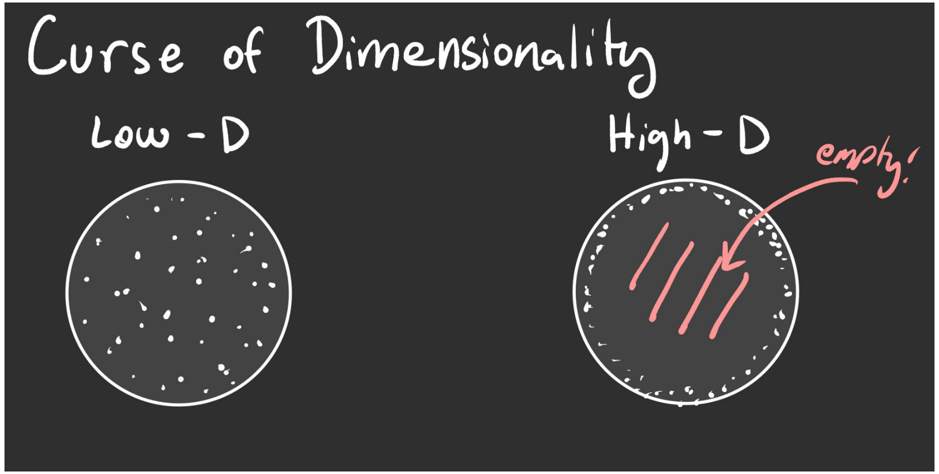
Properties of Nearest Neighbors

Pros

- No training required
- Learns complex, nonlinear functions

Cons

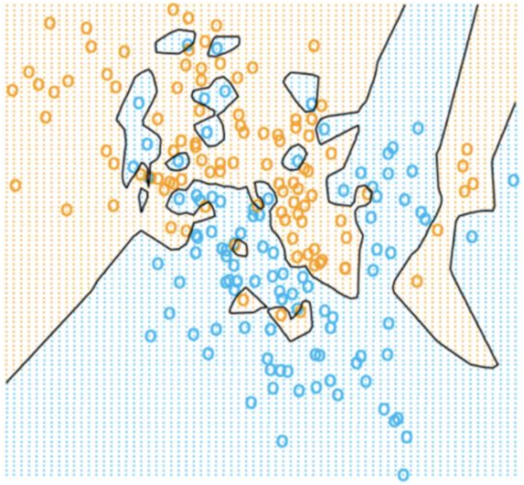
- High storage cost
- Slow at inference
- Curse of dimensionality: worse in higher dimensional data



Practice Question

Fall 2022 Final, 1.28

Which of these classifiers could have generated this decision boundary?



○ 15-NN (15 nearest neighbors)

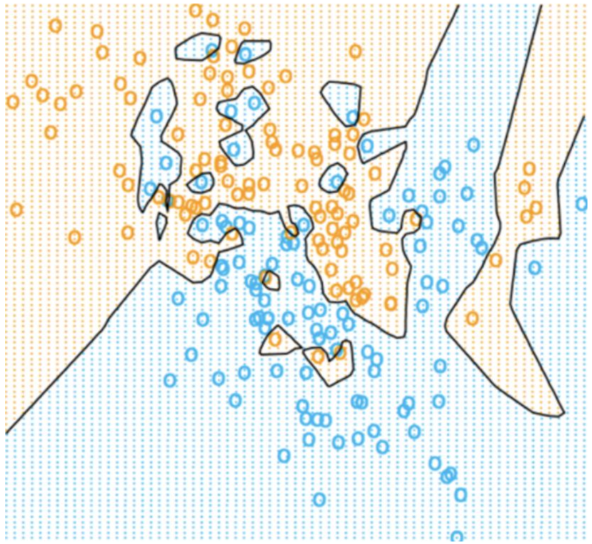
○ 1-NN (1 nearest neighbors)

○ Logistic Regression

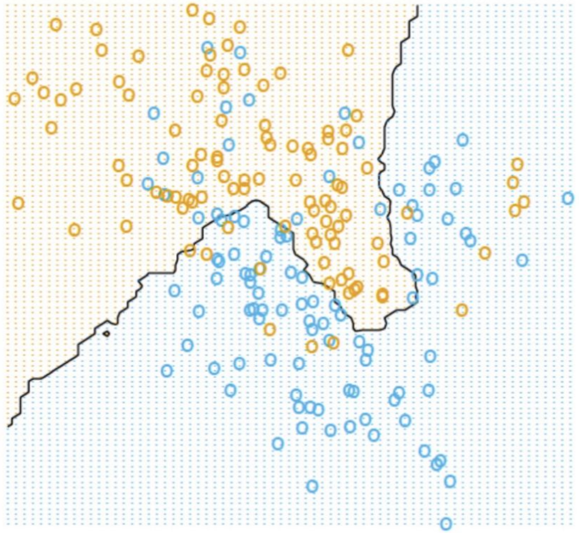
○ None of the above

Solution

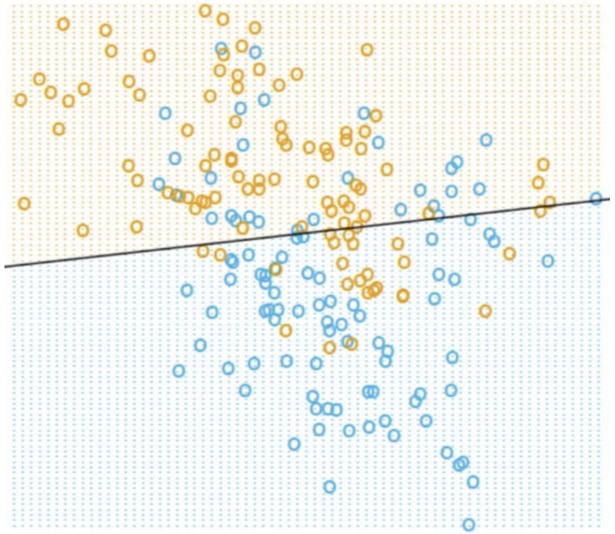
1-NN



15-NN



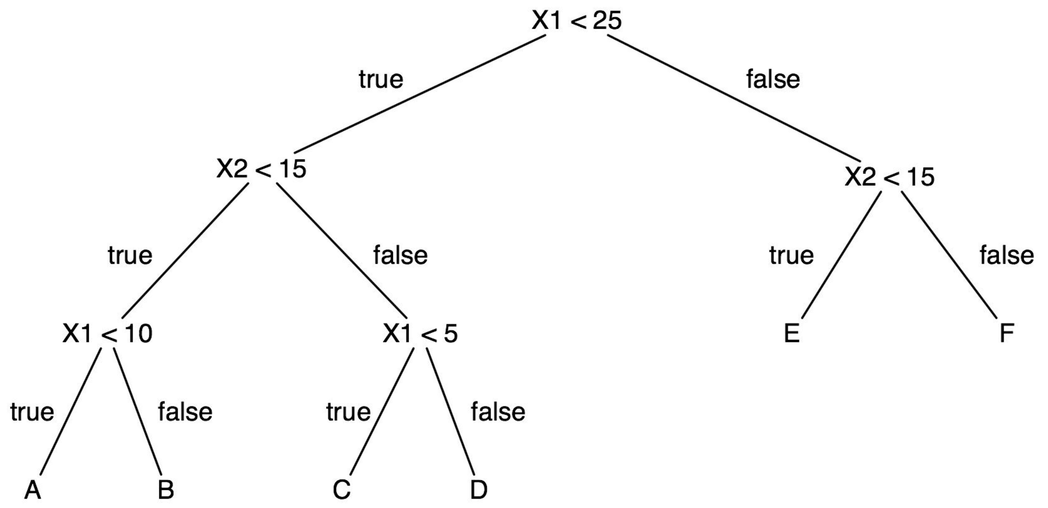
Logistic Regression



Decision Trees

Decision Trees

- At each node, split by a single feature
- Traverse down tree until you hit a leaf node, which is the output



Learning Decision Trees

- Greedy algorithm:
  - Start with empty tree
  - For each node:
    - If stopping condition reached:
      - Leaf label = average of data at that node
    - Else:
      - Split by **next best attribute**
      - Recurse to child nodes
- Next best attribute
  - Commonly: feature and split that maximizes **Information Gain**

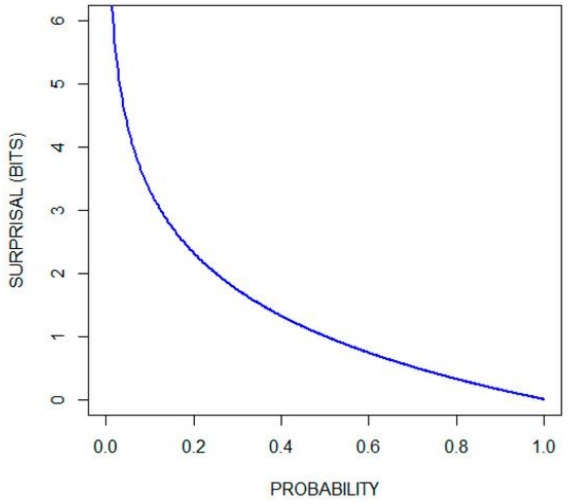
Entropy

- Entropy of a distribution: expected “surprise”

$$H(Y) \equiv E_y[-\log_2 P(Y)] = -\sum_k P(Y=k)\log_2 P(Y=k)$$

- Surprise:

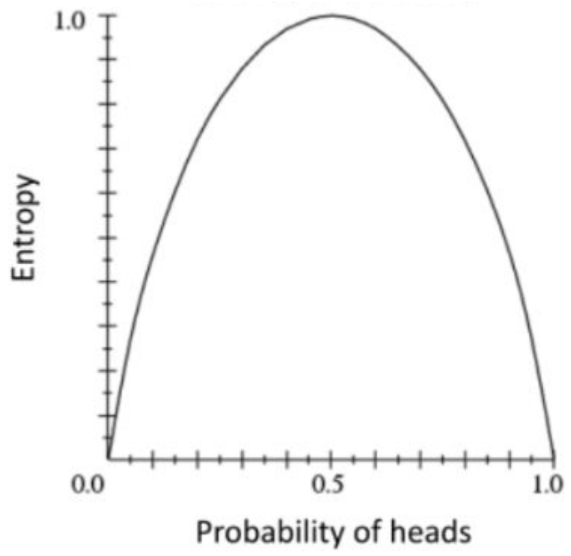
$$log\frac{1}{P(Y=k)} = -\log(P(Y=k))$$



Entropy

- Ex: entropy of a coin flip

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



Conditional Entropy and Information Gain

- Conditional Entropy: Expected entropy given random variable

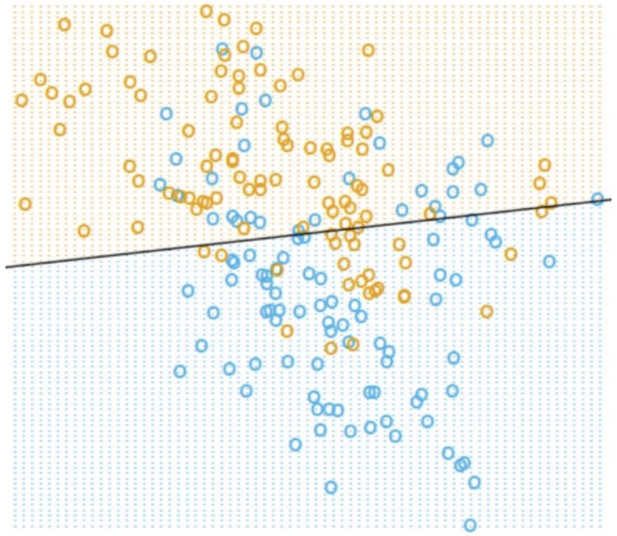
$$\mathrm{H}(Y|X) \equiv \sum_{x\in\mathcal{X}} p(x)\,\mathrm{H}(Y|X=x)$$

- Information Gain

$$I(X_{j,v};Y) := \underbrace{\boxed{H(Y)}}_{\text{constant}} - H(Y|X_{j,v})$$

Practice Questions

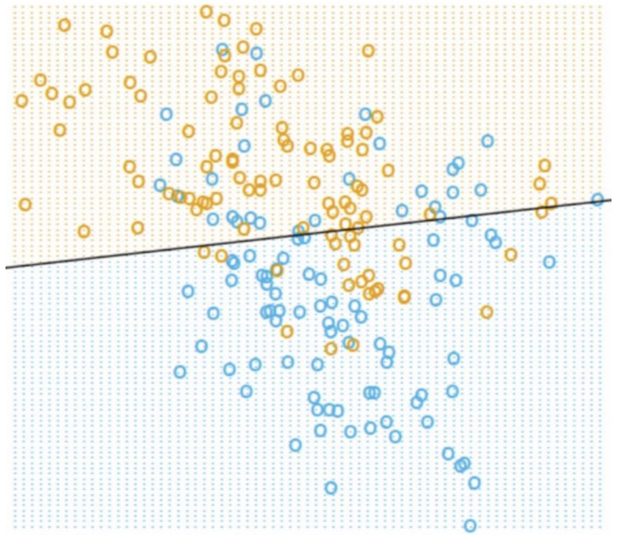
Q: Could this be a decision boundary created from a decision tree?



Solution

Q: Could this be a decision boundary created from a decision tree?

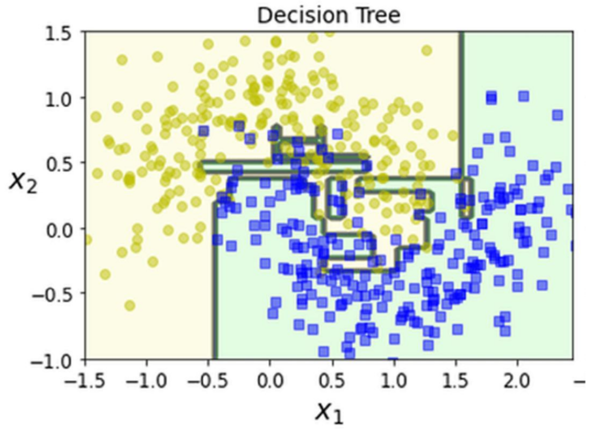
A: No, because decision trees create axis-aligned boundaries. Each node will only split on one feature

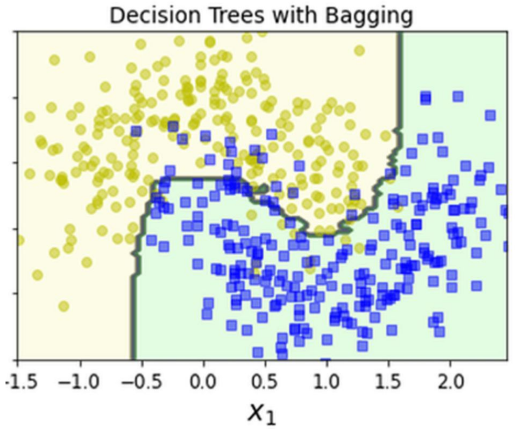


Ensembling

Bagging and Random Forests

- Decision trees can easily overfit. How can we reduce variance?
- Bagging (Bootstrap AGGregation)
  - Train M models, each with n' (usually n'=n) samples, sampled with replacement
  - Average M predictions to get bagged prediction
- Random Forests
  - Same as bagging, except at each split, choose only a random subset p' (usually p'=sqrt(p)) of features to split on





Boosting

- For bagging and random forests, we average the results from each model

$$y = \frac{1}{m} \sum_{m=1}^{M} G_m(x)$$

- However we can also consider using a weighted average

$$y = \sum_{m=1}^{M} \alpha_m G_m(x)$$

- Boosting algorithm:
  - Train next model conditioned on all previous models and their weights
  - Reweight models to minimize loss
  - Repeat
- Intuition behind boosting: reweighting of training points to emphasize those not currently correctly classified

Practice Question

(d) Is a random forest of stumps (trees with a single feature split or height 1) a good idea in general? Does the performance of a random forest of stumps depend much on the number of trees? Think about the bias of each individual tree and the bias of the average of all these random stumps.

Solution

(d) Is a random forest of stumps (trees with a single feature split or height 1) a good idea in general? Does the performance of a random forest of stumps depend much on the number of trees? Think about the bias of each individual tree and the bias of the average of all these random stumps.

Solution: Stumps generally have high bias; they are very simple models that cannot fit to anything with reasonable complexity. If we treat $\{Z_i\}$ as the set of possibly correlated predictions the stumps produce,

$$\mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \mu_z.$$

This tells us if each stump has high bias, averaging the predictions of all stumps will not reduce this bias. Thus a random forest of stumps is generally a bad idea no matter how many stumps we have.

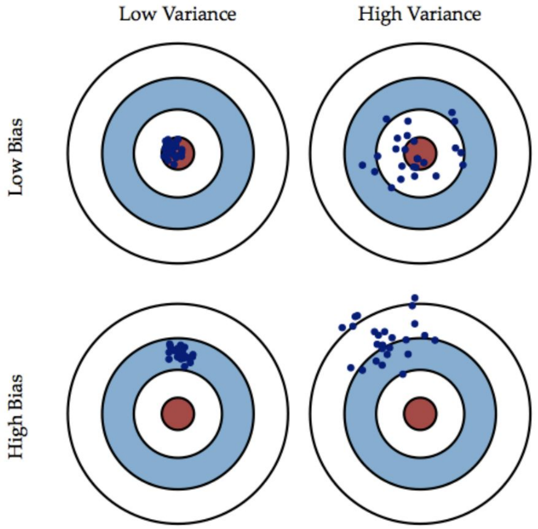
Bias-Variance

Bias-variance tradeoff

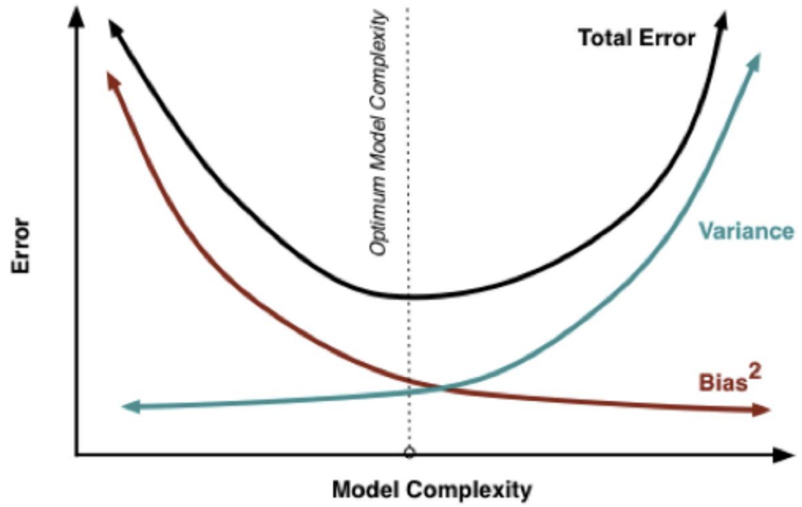
- Model error can be decomposed into three components

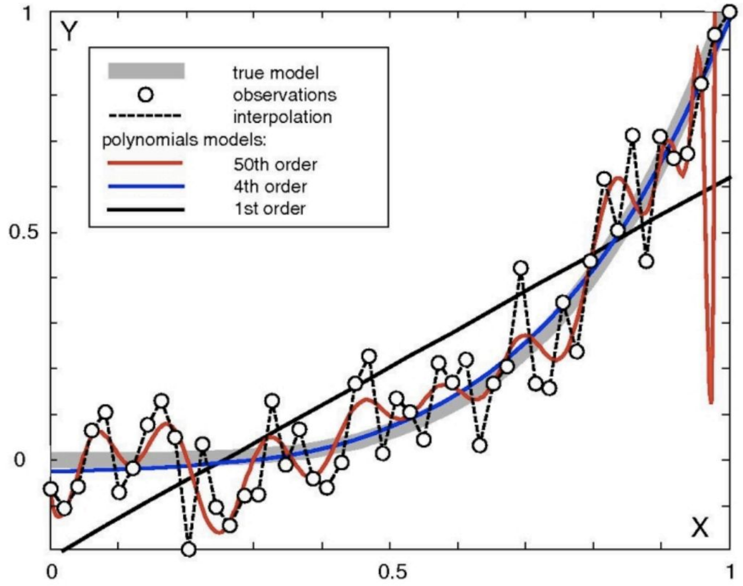
$$\varepsilon(\mathbf{x}; h) = \underbrace{\left(\mathbb{E}[h(\mathbf{x}; \mathcal{D})] - f(\mathbf{x})\right)^2}_{bias^2 \text{ of method}} + \underbrace{\mathrm{Var}(h(\mathbf{x}; \mathcal{D}))}_{\text{variance of method}} + \underbrace{\mathrm{Var}(Z)}_{\text{irreducible error}}$$

- **Bias**: measure of average difference between model output and ground truth over all possible training sets
- **Variance**: variance of model output over all possible training sets
- **Irreducible error**: error in model that cannot be controlled or eliminated



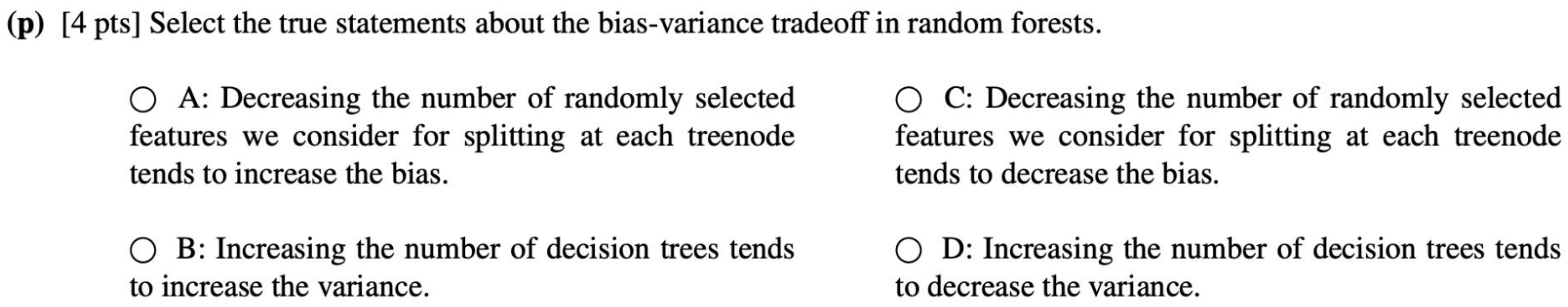
Bias-variance tradeoff





Practice Question

Spring 2023 Final, Q1(p)



(p) [4 pts] Select the true statements about the bias-variance tradeoff in random forests.

○ A: Decreasing the number of randomly selected features we consider for splitting at each treenode tends to increase the bias.

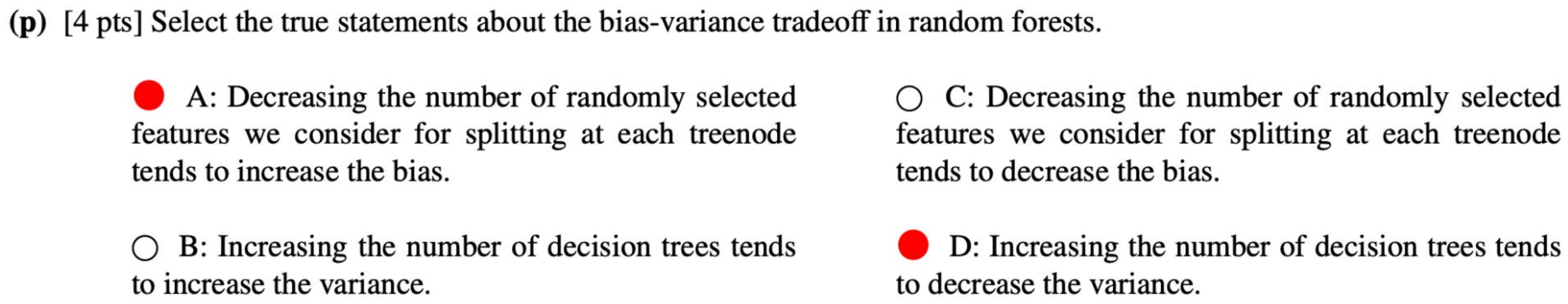
○ B: Increasing the number of decision trees tends to increase the variance.

○ C: Decreasing the number of randomly selected features we consider for splitting at each treenode tends to decrease the bias.

○ D: Increasing the number of decision trees tends to decrease the variance.

Solution

Since we are averaging over models, bias stays the same, but variance decreases



(p) [4 pts] Select the true statements about the bias-variance tradeoff in random forests.

[✓ A]: Decreasing the number of randomly selected features we consider for splitting at each treenode tends to increase the bias.

○ B: Increasing the number of decision trees tends to increase the variance.

○ C: Decreasing the number of randomly selected features we consider for splitting at each treenode tends to decrease the bias.

[✓ D]: Increasing the number of decision trees tends to decrease the variance.

Hidden Markov Models

Markov Models

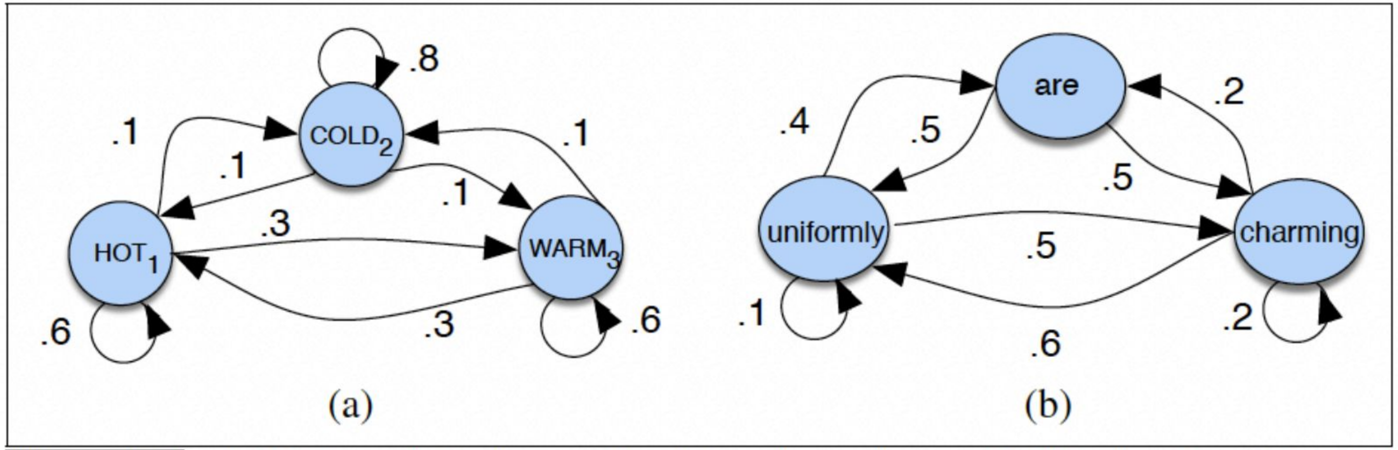
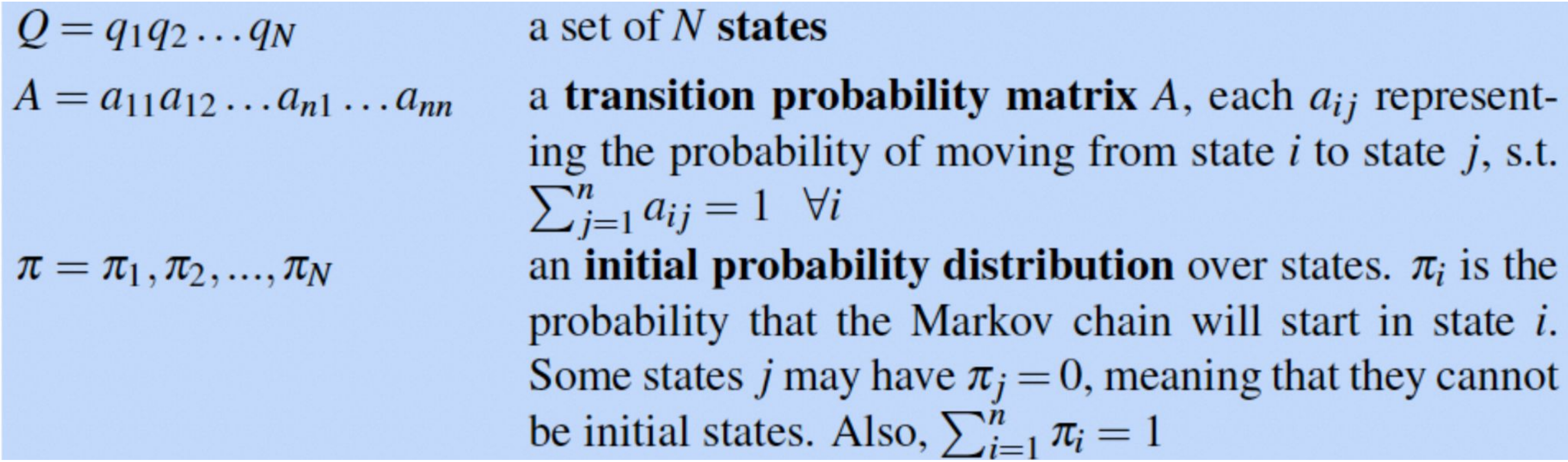
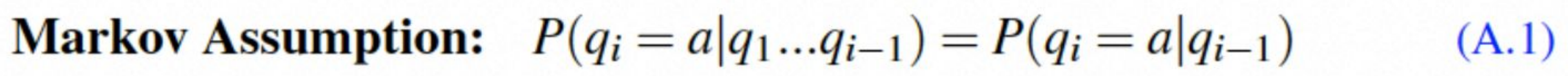


Figure A.1 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Markov Models

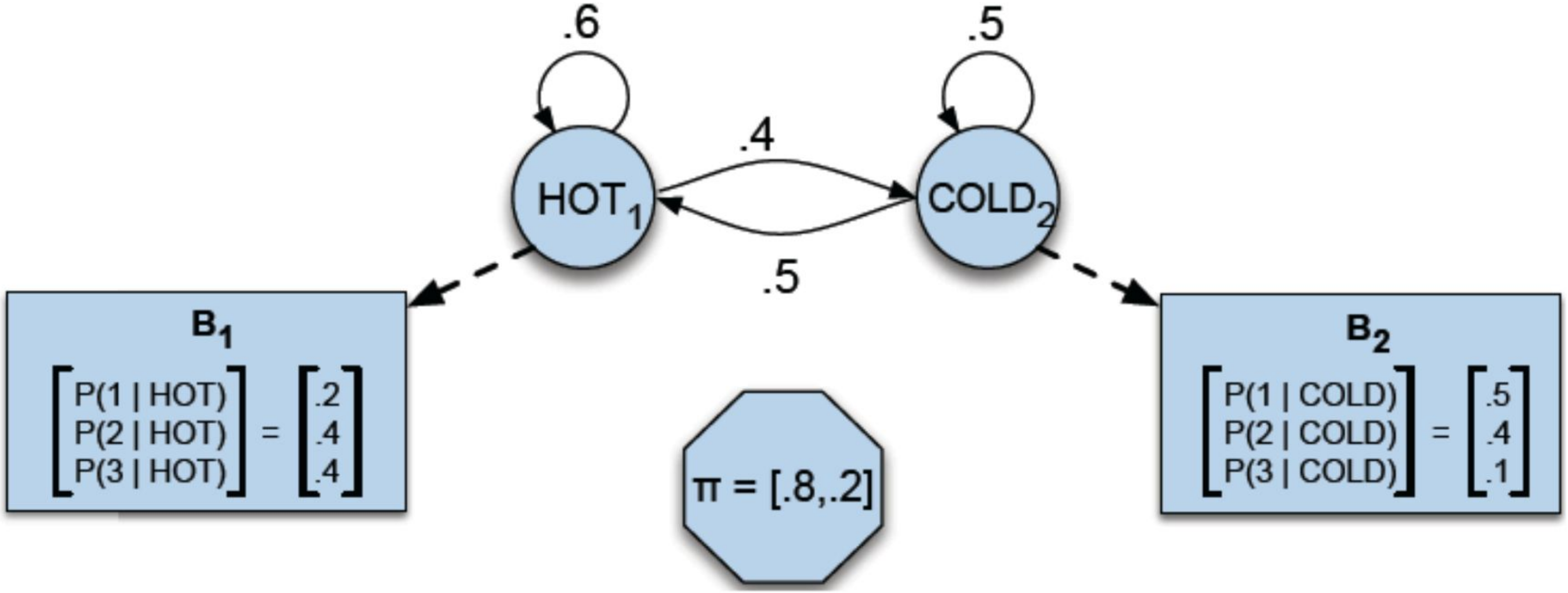


$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

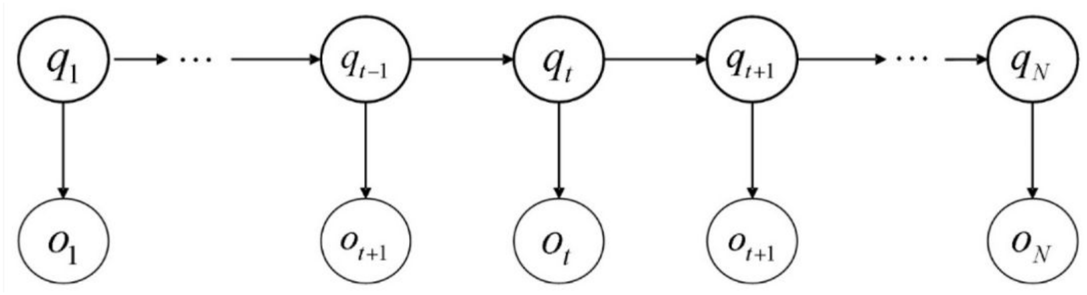


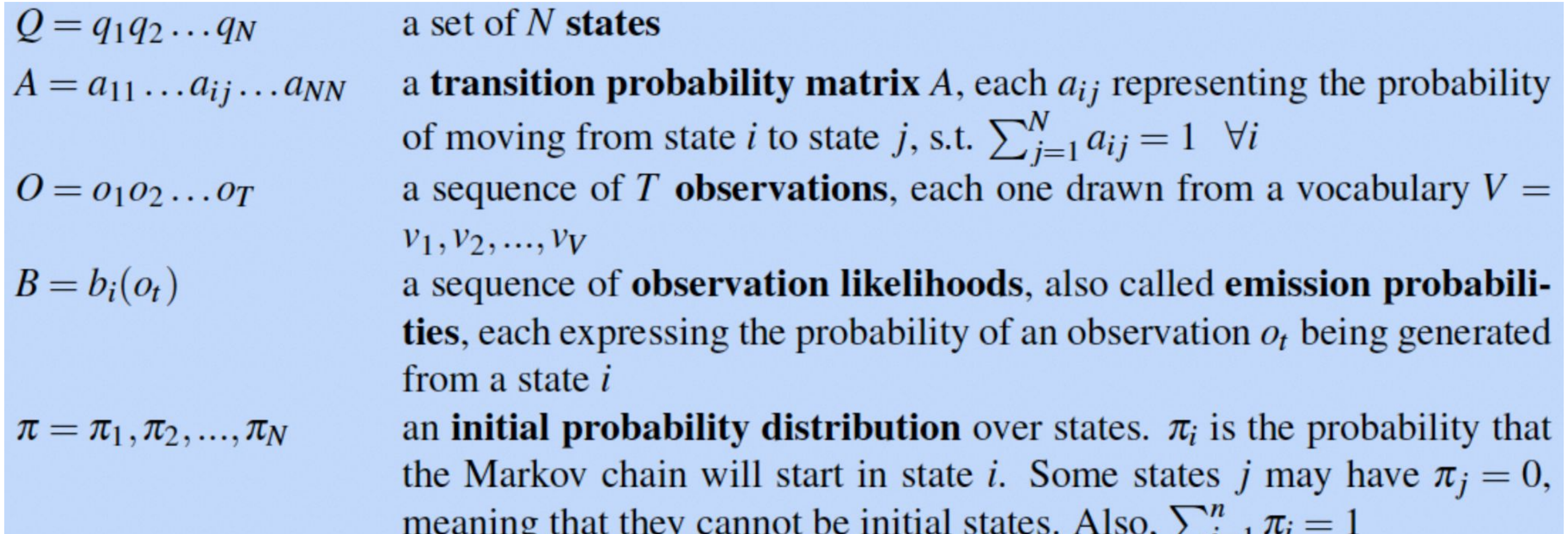
Markov Assumption: $P(q_i = a | q_1 \ldots q_{i-1}) = P(q_i = a | q_{i-1})$ (A.1)

Hidden Markov Models


.6
.5
.4
HOT$_1$
COLD$_2$
.5
B$_1$
$\begin{bmatrix} P(1 \mid HOT) \\ P(2 \mid HOT) \\ P(3 \mid HOT) \end{bmatrix} = \begin{bmatrix} .2 \\ .4 \\ .4 \end{bmatrix}$
$\pi = [.8,.2]$
B$_2$
$\begin{bmatrix} P(1 \mid COLD) \\ P(2 \mid COLD) \\ P(3 \mid COLD) \end{bmatrix} = \begin{bmatrix} .5 \\ .4 \\ .1 \end{bmatrix}$

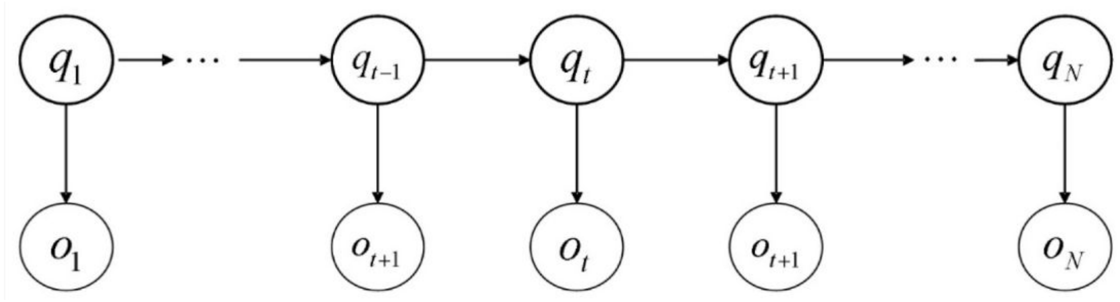
Hidden Markov Models





$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

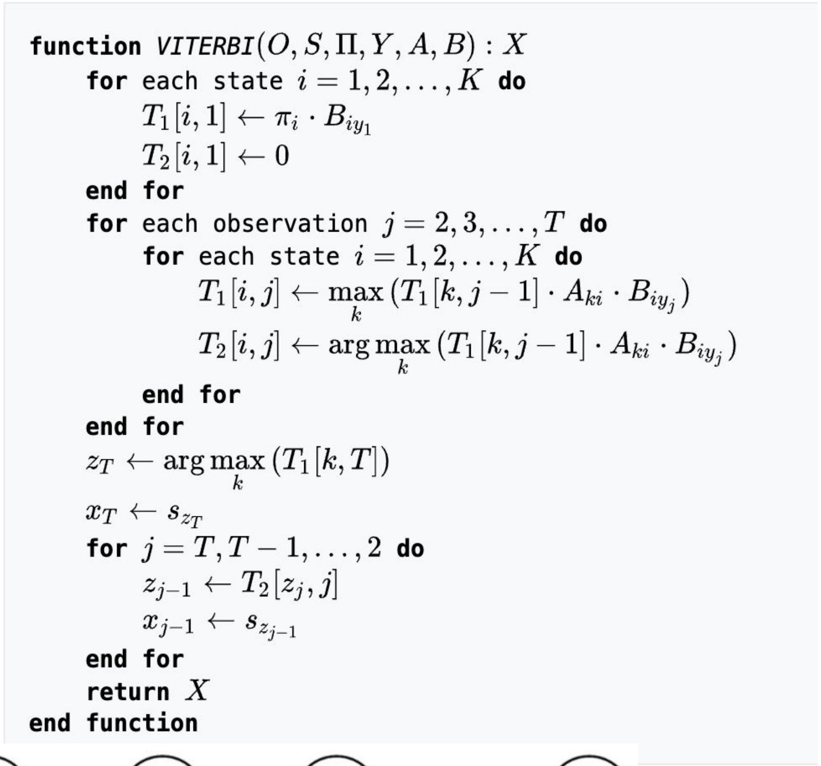
HMMs: Problems



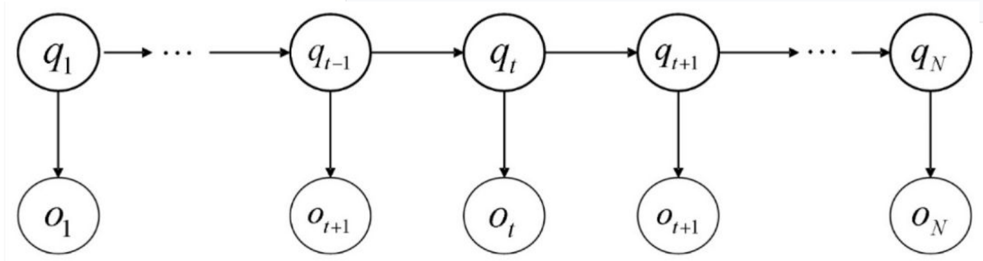
1. **Likelihood**:
   a. Given a specified HMM (transition probs, emission probs), compute the likelihood of an observation sequence O.
2. **Decoding**
   a. Given an HMM, find the best sequences of hidden states.
      i. Viterbi Algorithm
      ii. A worked out example: https://www.cis.upenn.edu/~cis2620/notes/Example-Viterbi-DNA.pdf
3. **Learning**
   a. Learn HMM parameters (transition and emission probs) from the observation sequence O.

Viterbi Pseudocode

- T1 stores prob of most likely path so far ending in state i.
- T2 stores the most recent observation in this path.
- We populate these matrices, computing a distribution over states at each timestep.
- Finally, we find the most likely path by working backwards from the final state.



```
function VITERBI$(O, S, \Pi, Y, A, B) : X$
    for each state $i = 1, 2, \ldots, K$ do
        $T_1[i, 1] \leftarrow \pi_i \cdot B_{iy_1}$
        $T_2[i, 1] \leftarrow 0$
    end for
    for each observation $j = 2, 3, \ldots, T$ do
        for each state $i = 1, 2, \ldots, K$ do
            $T_1[i, j] \leftarrow \max_k (T_1[k, j-1] \cdot A_{ki} \cdot B_{iy_j})$
            $T_2[i, j] \leftarrow \arg\max_k (T_1[k, j-1] \cdot A_{ki} \cdot B_{iy_j})$
        end for
    end for
    $z_T \leftarrow \arg\max_k (T_1[k, T])$
    $x_T \leftarrow s_{z_T}$
    for $j = T, T-1, \ldots, 2$ do
        $z_{j-1} \leftarrow T_2[z_j, j]$
        $x_{j-1} \leftarrow s_{z_{j-1}}$
    end for
    return $X$
end function
```



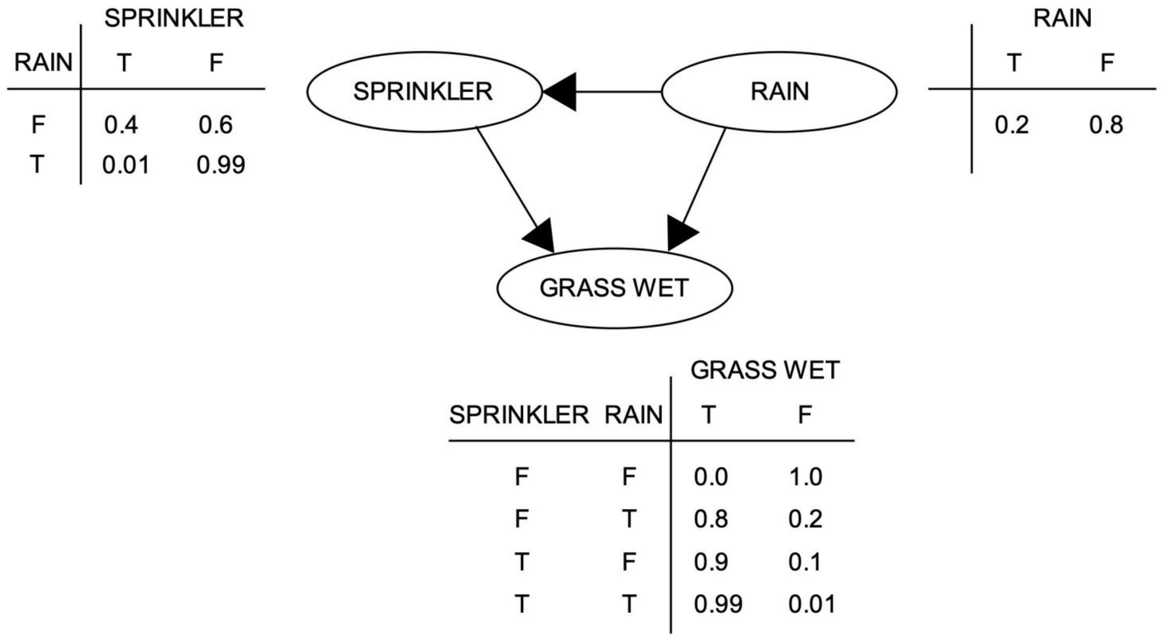
Things to understand

- In what sense is this optimal and can you prove that it's optimal?
    - Computes the most likely path.
- Why do we only need to store the most recent states x_{j-1}?
    - The Markov Property
- Why do we go backwards to find the path?
    - Because T1 stores the probability of the most likely path ending in state i.

Probabilistic Graphical Models

Probabilistic Graphical Models

- A graph where each node represents some random variable and edges represent dependence relationships
- DAGs help us achieve tractability through conditional independence



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

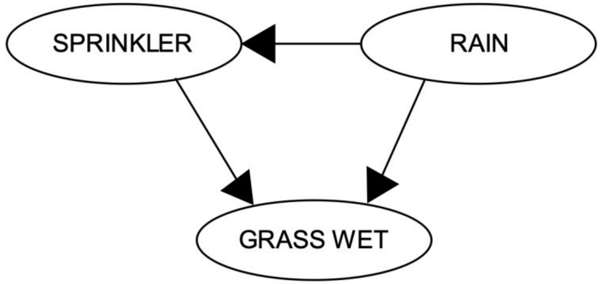
PGMs: Problems

1. Factorization and Probability Calculations
    a. Factoring the joint density based on the links in the graph and answering questions about conditional independence (d-separation) and conditional probabilities
2. State estimation
    a. Same as HMMs
3. Reformulating HMMs as PGMs
    a. Turn an HMM into a DAG

Practice Problem

Given that the grass is wet (G), what is the probability that it rained (R)?

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Solution

$$\Pr(R=T \mid G=T) = \frac{\Pr(G=T, R=T)}{\Pr(G=T)} = \frac{\sum_{x\in\{T,F\}} \Pr(G=T, S=x, R=T)}{\sum_{x,y\in\{T,F\}} \Pr(G=T, S=x, R=y)}$$

We can calculate the probability of any case using the joint probability distribution e.g.

$$\begin{aligned}\Pr(G=T, S=T, R=T) &= \Pr(G=T \mid S=T, R=T)\Pr(S=T \mid R=T)\Pr(R=T)\\ &= 0.99 \times 0.01 \times 0.2\\ &= 0.00198.\end{aligned}$$

Then the numerical results (subscripted by the associated variable values) are

$$\Pr(R=T \mid G=T) = \frac{0.00198_{TTT} + 0.1584_{TFT}}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0.0_{TFF}} = \frac{891}{2491} \approx 35.77\%.$$

Solution

Let R be the event that it rained, D be the event that the grass is dry, and S be the event that the sprinkler went off.

$P(R, D, S) = P(R)P(D|R, S)P(S|R)$

$P(R \mid D) = P(D \mid R)P(R) / P(D)$

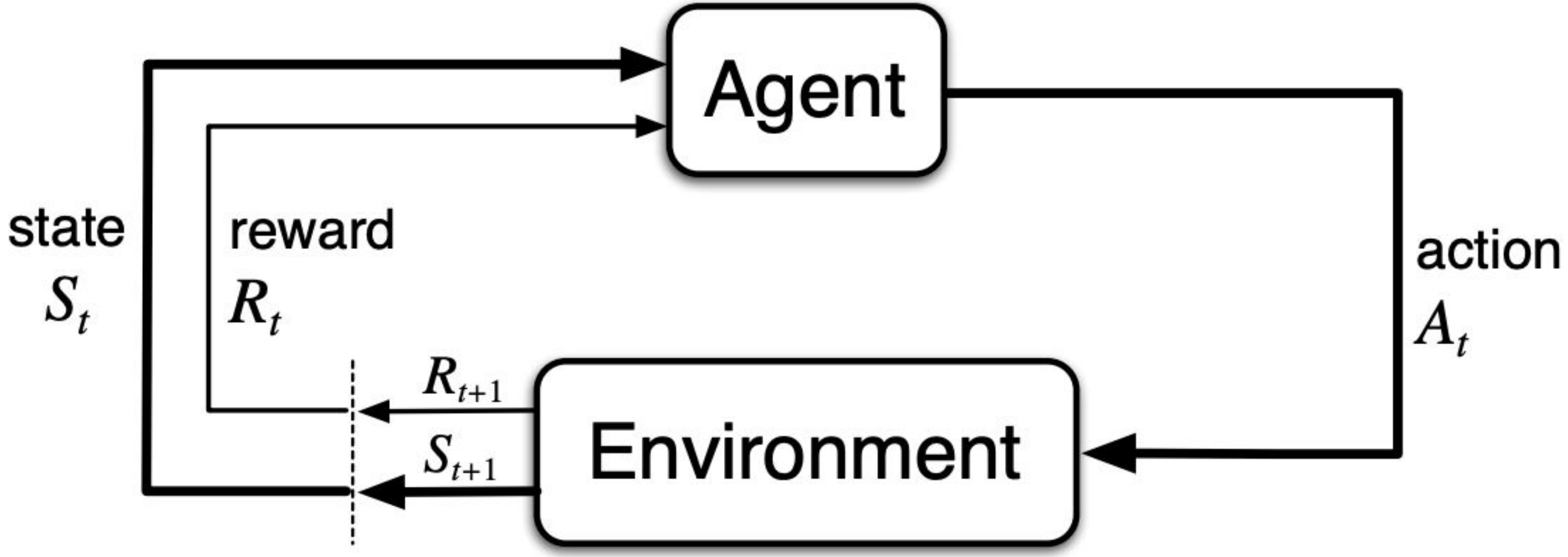
$P(D \mid R) = P(D| R, S)P(S|R)P(R) + P(D \mid R, \sim S)P(\sim S)P(R) =$

Practice Problems

- Notes from cs188: https://inst.eecs.berkeley.edu/~cs188/fa23/assets/notes/cs188-fa23-note13.pdf

Markov Decision Processes and RL

Markov Decision Process



Markov Decision Process

- Characterized by a state space S, policy π (and actions A), rewards R, and transition dynamics $P(S_t, R_t \mid S_{t-1}, A_{t-1})$
- MDPs satisfy the Markov property, ie conditioning on all history is equivalent to conditioning on just the previous state.
- We seek to learn policies that **maximize the sum of discounted rewards, or return.** By optimizing our policy subject to the uncertainty in the environment.

Definitions

Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

State Value function:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

Action-value function:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a]$$

Expectation is taken over our policy

The Bellman Equation

Value function as the expectation of the q function over the policy:

$$V_\pi(s) = E_\pi[q_\pi(s,\, a)] = \textstyle\sum_{a\in A} \pi(a|s)q(s,\, a)$$

Q function as the expectation of next-step value over transition dynamics

$$q_\pi(s,\, a) = E_\pi[R_t + \gamma V_\pi(S_{t+1})|\, S_t,\, A] = \sum_{s',r}\big(r + \gamma V_\pi(s')\big)p(s', r|s,\, a)$$

The Bellman Equation: a recursive definition of the value function

$$V_\pi(s) = \sum_{a\in A}\pi(a|s)\sum_{s',r}\big(r + \gamma V_\pi(s')\big)p(s', r|s,\, a)$$

Policy Iteration

1. Initialize value function and policy randomly
2. **Policy evaluation:** estimate the value function associated with the current policy using the Bellman equations (fixed point strategy).
3. **Policy improvement:** improve the current policy by leveraging the value function.
4. Go back to step 2 unless converged.

$$\text{Policy evaluation: } v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

$$\text{Policy improvement: } \pi'(s) = \operatorname*{argmax}_{a} \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Value Iteration

1. Initialize value function
2. Update value function
3. Repeat until convergence

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Value Iteration

1. Initialize value function
2. Update value function
3. Repeat until convergence

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_k(s')]$$

Once the algorithm has converged; how can we know which actions to take?

Example

You are controlling a spacecraft on a mission to explore and gather data from various celestial bodies in a solar system. The spacecraft can be in one of three states based on its energy levels: ’FullEnergy’, ’LowEnergy’, and ’Depleted’. ’Depleted’ is a terminal state, representing the spacecraft running out of energy and being unable to continue its mission. We denote the states as $\mathcal{S} = \{F, L, D\}$.

At each state (except ”Depleted”), there are two possible actions: ’Conserve’ and ’Explore’. ’Conserve’ represents cautious exploration with energy conservation, while ’Explore’ represents aggressive exploration consuming more energy. We denote the actions as $\mathcal{A} = \{C, E\}$.

Example Transition Dynamics

Entries of table specify the distribution of next states: [full, low, depleted] and reward

State / Action	Conserve	Explore
Full Energy	[1, 0, 0]: 1	[.5, .5, 0]: 2
Low Energy	[.5, .5, 0]: 1	[0, 0, 1]: -10
Depleted	[0, 0, 1]: 0	[0, 0, 1]: 0

$$p(s', r \mid s, a)$$

Policy Iteration

Suppose we initialize with a policy that always conserves regardless of the states, i.e. $\pi_0(C|s) = 1, \pi_0(E|s) = 0$ for all s. Also, we initialize value functions $v_0(s) = 0$ for all s. Let the discount rate $\gamma = 0.5$. Run policy iteration for two iterations. Does policy iteration converges after two iterations?

State / Action	Conserve	Explore
Full Energy	[1, 0, 0]: 1	[.5, .5, 0]: 2
Low Energy	[.5, .5, 0]: 1	[0, 0, 1]: -10
Depleted	[0, 0, 1]: 0	[0, 0, 1]: 0

Policy evaluation: $v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$

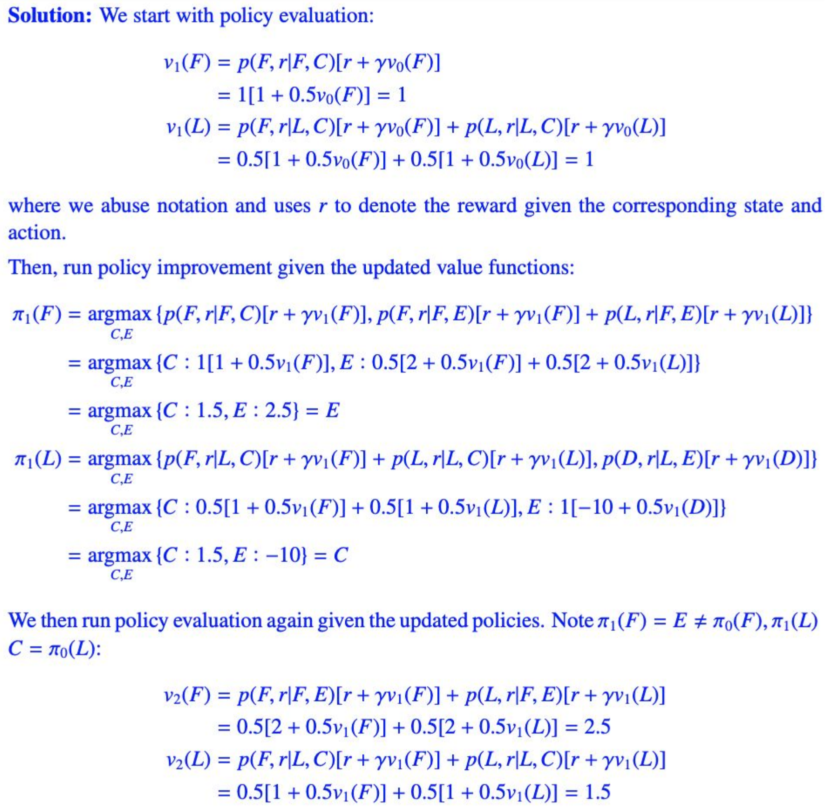
Policy improvement: $\pi'(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$

$$\text{Policy evaluation: } v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

Policy Iteration

$$\text{Policy improvement: } \pi'(s) = \underset{a}{\text{argmax}} \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

State / Action	Conserve	Explore
Full Energy	[1, 0, 0]: 1	[.5, .5, 0]: 2
Low Energy	[.5, .5, 0]: 1	[0, 0, 1]: -10
Depleted	[0, 0, 1]: 0	[0, 0, 1]: 0



Solution: We start with policy evaluation:

$$\begin{aligned} v_1(F) &= p(F,r|F,C)[r + \gamma v_0(F)] \\ &= 1[1 + 0.5v_0(F)] = 1 \\ v_1(L) &= p(F,r|L,C)[r + \gamma v_0(F)] + p(L,r|L,C)[r + \gamma v_0(L)] \\ &= 0.5[1 + 0.5v_0(F)] + 0.5[1 + 0.5v_0(L)] = 1 \end{aligned}$$

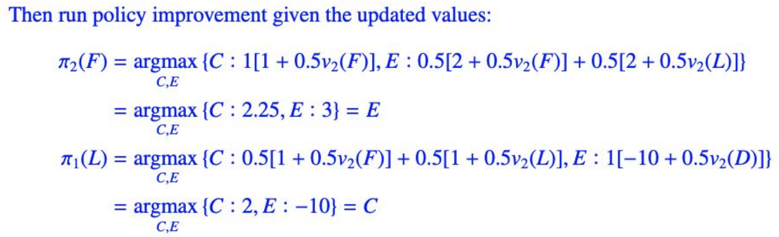
where we abuse notation and uses r to denote the reward given the corresponding state and action.

Then, run policy improvement given the updated value functions:

$$\begin{aligned} \pi_1(F) &= \underset{C,E}{\text{argmax}}\,\{p(F,r|F,C)[r + \gamma v_1(F)], p(F,r|F,E)[r + \gamma v_1(F)] + p(L,r|F,E)[r + \gamma v_1(L)]\} \\ &= \underset{C,E}{\text{argmax}}\,\{C : 1[1 + 0.5v_1(F)], E : 0.5[2 + 0.5v_1(F)] + 0.5[2 + 0.5v_1(L)]\} \\ &= \underset{C,E}{\text{argmax}}\,\{C : 1.5, E : 2.5\} = E \\ \pi_1(L) &= \underset{C,E}{\text{argmax}}\,\{p(F,r|L,C)[r + \gamma v_1(F)] + p(L,r|L,C)[r + \gamma v_1(L)], p(D,r|L,E)[r + \gamma v_1(D)]\} \\ &= \underset{C,E}{\text{argmax}}\,\{C : 0.5[1 + 0.5v_1(F)] + 0.5[1 + 0.5v_1(L)], E : 1[-10 + 0.5v_1(D)]\} \\ &= \underset{C,E}{\text{argmax}}\,\{C : 1.5, E : -10\} = C \end{aligned}$$

We then run policy evaluation again given the updated policies. Note $\pi_1(F) = E \neq \pi_0(F), \pi_1(L)$ $C = \pi_0(L)$:

$$\begin{aligned} v_2(F) &= p(F,r|F,E)[r + \gamma v_1(F)] + p(L,r|F,E)[r + \gamma v_1(L)] \\ &= 0.5[2 + 0.5v_1(F)] + 0.5[2 + 0.5v_1(L)] = 2.5 \\ v_2(L) &= p(F,r|L,C)[r + \gamma v_1(F)] + p(L,r|L,C)[r + \gamma v_1(L)] \\ &= 0.5[1 + 0.5v_1(F)] + 0.5[1 + 0.5v_1(L)] = 1.5 \end{aligned}$$



Then run policy improvement given the updated values:

$$\begin{aligned} \pi_2(F) &= \underset{C,E}{\text{argmax}}\,\{C : 1[1 + 0.5v_2(F)], E : 0.5[2 + 0.5v_2(F)] + 0.5[2 + 0.5v_2(L)]\} \\ &= \underset{C,E}{\text{argmax}}\,\{C : 2.25, E : 3\} = E \\ \pi_1(L) &= \underset{C,E}{\text{argmax}}\,\{C : 0.5[1 + 0.5v_2(F)] + 0.5[1 + 0.5v_2(L)], E : 1[-10 + 0.5v_2(D)]\} \\ &= \underset{C,E}{\text{argmax}}\,\{C : 2, E : -10\} = C \end{aligned}$$

Value Iteration

Run value iteration for two iterations. Does it converge after two iterations?

State / Action	Conserve	Explore
Full Energy	[1, 0, 0]: 1	[.5, .5, 0]: 2
Low Energy	[.5, .5, 0]: 1	[0, 0, 1]: -10
Depleted	[0, 0, 1]: 0	[0, 0, 1]: 0

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

State / Action	Conserve	Explore
Full Energy	[1, 0, 0]: 1	[.5, .5, 0]: 2
Low Energy	[.5, .5, 0]: 1	[0, 0, 1]: -10
Depleted	[0, 0, 1]: 0	[0, 0, 1]: 0

Value Iteration

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Solution: By definition of value iterations:

$$\begin{aligned} v_1(F) &= \max\{p(F, r|F, C)[r + \gamma v_0(F)], p(F, r|F, E)[r + \gamma v_0(F)] + p(L, r|F, E)[r + \gamma v_0(L)]\} \\ &= \max\{1[1 + 0.5v_0(F)], 0.5[2 + 0.5v_0(F)] + 0.5[2 + 0.5v_0(L)]\} \\ &= \max\{1, 2\} = 2 \\ v_1(L) &= \max\{p(F, r|L, C)[r + \gamma v_0(F)] + p(L, r|L, C)[r + \gamma v_0(L)], p(D, r|L, E)[r + \gamma v_0(D)]\} \\ &= \max\{0.5[1 + 0.5v_0(F)] + 0.5[1 + 0.5v_0(L)], 1[-10 + 0.5v_0(D)]\} \\ &= \max\{1, -10\} = 1 \end{aligned}$$

Using these updated values, we can run another step of value iteration:

$$\begin{aligned} v_2(F) &= \max\{p(F, r|F, C)[r + \gamma v_1(F)], p(F, r|F, E)[r + \gamma v_1(F)] + p(L, r|F, E)[r + \gamma v_1(L)]\} \\ &= \max\{1[1 + 0.5v_1(F)], 0.5[2 + 0.5v_1(F)] + 0.5[2 + 0.5v_1(L)]\} \\ &= \max\{2, 2.75\} = 2.75 \\ v_2(L) &= \max\{p(F, r|L, C)[r + \gamma v_1(F)] + p(L, r|L, C)[r + \gamma v_1(L)], p(D, r|L, E)[r + \gamma v_1(D)]\} \\ &= \max\{0.5[1 + 0.5v_1(F)] + 0.5[1 + 0.5v_1(L)], 1[-10 + 0.5v_1(D)]\} \\ &= \max\{1.75, -10\} = 1.75 \end{aligned}$$

After two rounds, value iteration hasn't converged yet.

Robotics/Language/Vision

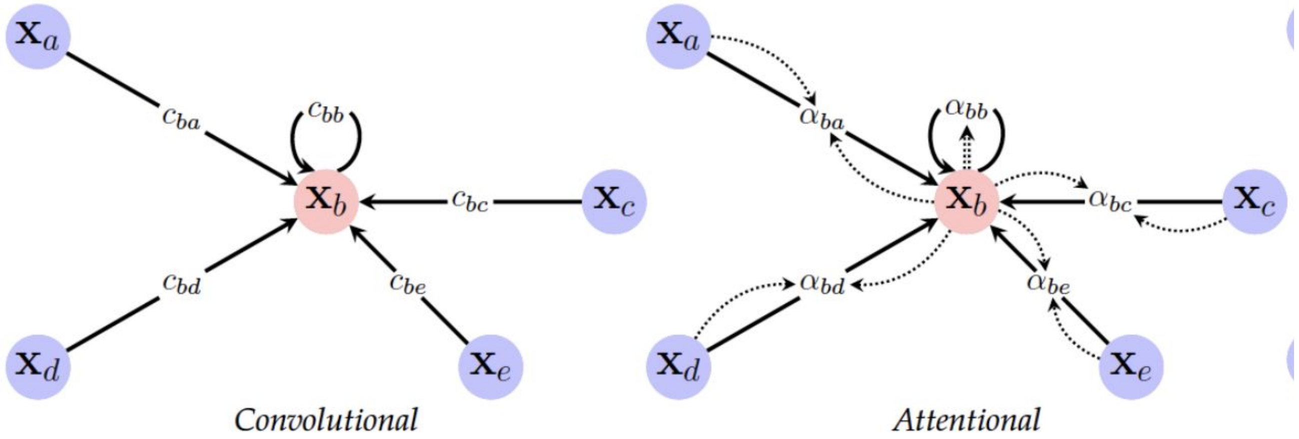
Graph Neural Networks

Graph Neural Networks

- A graph is defined on a set of nodes V with edges E.
- The primary mechanism in GNNs is **message passing**

$$\begin{aligned}\mathbf{h}_u^{(k)} &= \phi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \bigoplus\left(\left\{\overset{\text{Message function}}{\psi^{(k)}}(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}), \forall v \in \mathcal{N}(u)\right\}\right)\right) \\ &= \phi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \underset{\text{AGGREGATE}}{\bigoplus}\left(\left\{\mathbf{m}_{vu}^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right)\right) \\ &= \underset{\text{COMBINE}}{\phi^{(k)}}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_u^{(k)}\right)\end{aligned}$$

Flavors of Message Passing



$$h_u^{(k)} = \phi\left(h_u^{(k-1)}, \bigoplus\{W^k h_v^{(k-1)} \mid v \in \mathcal{N}(u)\}\right)$$

$$h_u^{(k)} = \phi\left(h_u^{(k-1)}, \bigoplus\left\{a(h_v^{(k-1)}, h_u^{(k-1)})h_v^{(k-1)}) \mid v \in \mathcal{N}(u)\right\}\right)$$

Practice Question

1. How many parameters do we have in a GNN with the following update function?

$$h_u^{(k)} = \sigma\left(W_0^{(k)} h_u^{(k-1)} + \sum_i W_1^{(k)} h_{v_i}\right),\ W_i^{(k)} \in \mathbb{R}^{d\times k}$$

2. What about a CNN with kernel size *k* x *k* and *m* input channels and *n* output channels?

Solution

1. $2dk$
2. k^2mn

Note that neither answer depends on $|V|$.

Tasks

Convolutional Neural Networks (CNNs)	Graph Neural Networks (GNNs)
Image-level tasks (Classification or Regression Tasks) (One output / target for entire image. Ex: dog, cat, etc.)	**Graph-level tasks** (Classification Tasks. Ex: Graph of a particular molecule: deciding if it is poisonous or not? Or at what temperature will it melt?)
Pixel-level tasks (Ex: Semantic segmentation for classification of every pixel)	**Node/Edge-level tasks** (Ex: Graph of customers and products in commercial data deciding the pricing of products or how to give) recommendations for each customer)

Geometric Learning: In/Equivariances

- In Graphs neighbors have no order, so aggregation functions must be **permutation invariant.**
  - Mean the arguments could be permutated, but the result should be the same ie $f(PA) = f(A)$ for a permutation matrix P.
  - This is a general property of GNNs.
- We can also induce translational in/equivariance
  - Think of translational equivariance in convolutional layers and *approximate* invariance induced by pooling operations.
- Other kinds of invariance
  - Rotational, flipping, perspective shift.
  - A general technique to induce approximate invariance is data augmentation

Geometric Learning: In/Equivariances

- In graphs, neighbors have no order, so aggregation functions must be permutation **invariant.**
    - Mean the arguments could be permuted, but the result should be the same.
    - That is, $f(PA) = f(A)$ for a permutation matrix P.
- Making the aggregation function permutation **invariant** results in the graph neural network being permutation **equivariant**.
    - Means that permutations of the arguments results in the same permutation of the outputs.
    - That is, $f(PA) = Pf(A)$ for a permutation matrix P.

Practice Question

Which of the following are permutation-invariant aggregation functions?

1. $f(x,\ y,\ z) = e^{2x+3y+z}$
2. $f(x,\ y, z) = xyz^2$
3. $f(x, y, z) = \max(x+y,\ y+z,\ \min(x,\ y,\ z))$
4. $f(x, y, z) = \min(x+y,\ x+z,\ y+z,\ 2x,\ 2z,\ 2y)$

Practice Question: Solution

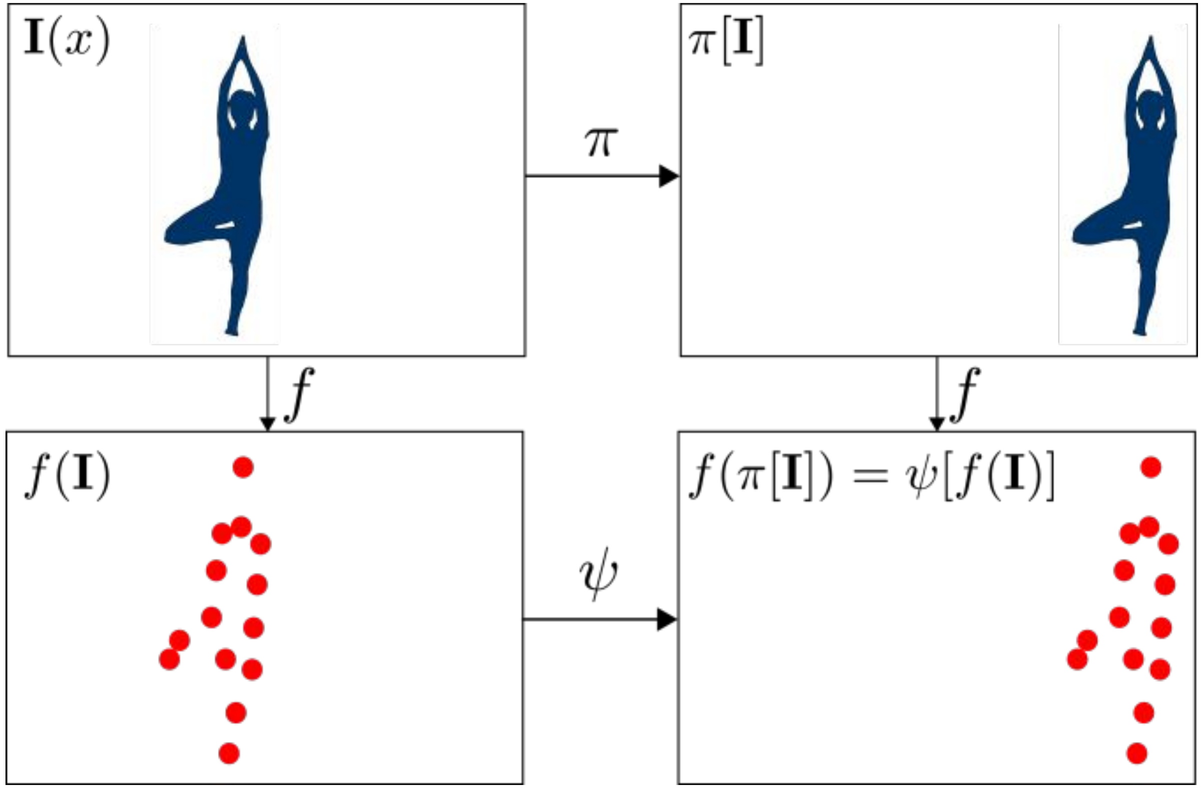
Which of the following are permutation-invariant aggregation functions?

1. $f(x, y, z) = e^{2x+3y+z}$
2. $f(x, y, z) = xyz^2$
3. $f(x, y, z) = \max(x + y, y + z, \min(x, y, z))$
4. $\boxed{f(x, y, z) = \min(x + y, x + z, y + z, 2x, 2z, 2y)}$

4 is the only permutation invariant function

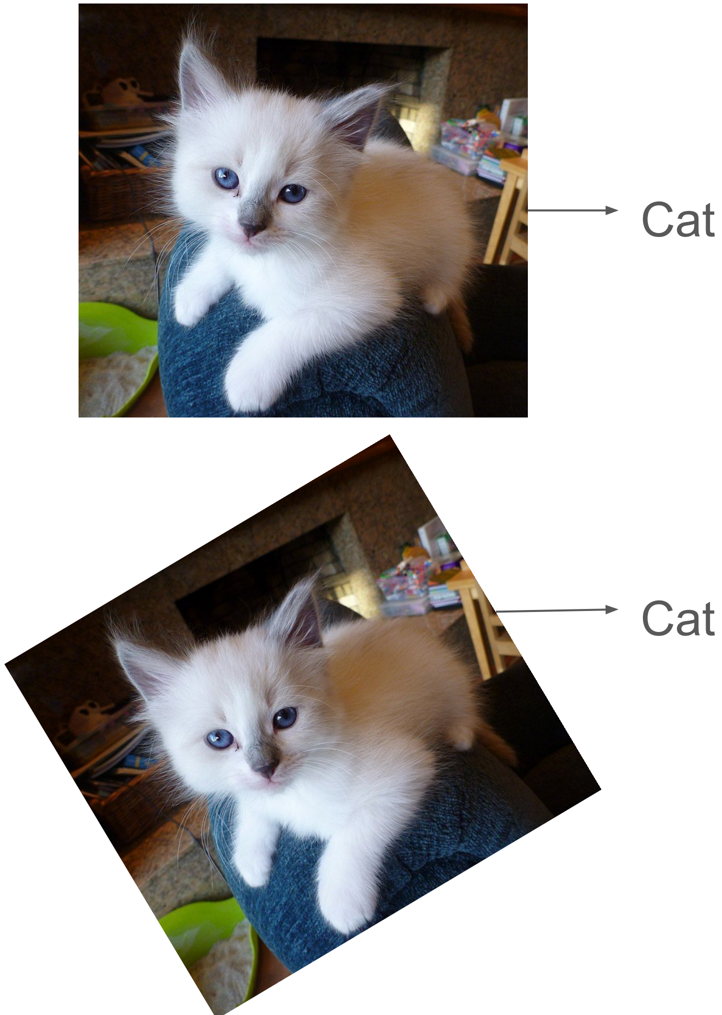
Translational Equivariance

- Useful for pixel and node-level tasks
- Ex: semantic segmentation or node classification



Rotational Invariance

- Useful for graph and image-level tasks
- Ex: molecule classification or image classification



Practice Question

In the following scenarios, would we want invariance or equivariance with respect to rotation?

1. Estimating the pose (x, y, z, orientation) of a chair in a scene.
2. Classifying an image into [cat, dog].
3. Predicting if a crystal structure would be stable given a molecular representation.
4. Predicting whether each pixel in an image belongs to a certain class.

Practice Question

In the following scenarios, would we want invariance or equivariance with respect to rotation?

1. Estimating the pose (x, y, z, orientation) of a chair in a scene.
   a. **Equivariance**: if the chair moves, we'd want to reflect this in the output
2. Classifying an image into [cat, dog].
   a. **Invariance**: a rotated cat is still a cat
3. Predicting if a crystal structure would be stable given a molecular representation.
   a. **Invariance**: if a molecule is stable, it should be stable when viewed from a different orientation
4. Predicting whether each pixel in an image is a pixel of a cat.
   a. **Equivariance**: if a cat in the image is rotation, the prediction of the pixels corresponding to that cat should too

Langevin MCMC

Score-based generative models

Class of generative models that learn an approximation to the score

$$s_\theta(x) \approx \nabla_x \log p_{\text{data}}(x)$$

This choice is particularly convenient to generate new samples, using Langevin dynamics:

$$x_{t+1} = x_t + \eta \nabla_x \log p_{\text{data}}(x_t) + \sqrt{2\eta} z_t \quad \text{where} \quad z_t \sim \mathcal{N}(0, I)$$

Strategies to learn score-based generative models

1. *Maximum likelihood:* $$\min_\theta \mathbb{E}_{p_{\text{data}}}\left[\log p_\theta(x)\right]$$

2. *Score matching:* $$\min_\theta \mathbb{E}_{p_{\text{data}}}\left[\|\nabla_x \log p(x) - s_\theta(x)\|^2\right]$$

What are the limitations of these approaches?

Strategies to learn score-based generative models

1. *Maximum likelihood:* $$\min_\theta \mathbb{E}_{p_{\text{data}}} [\log p_\theta(x)]$$

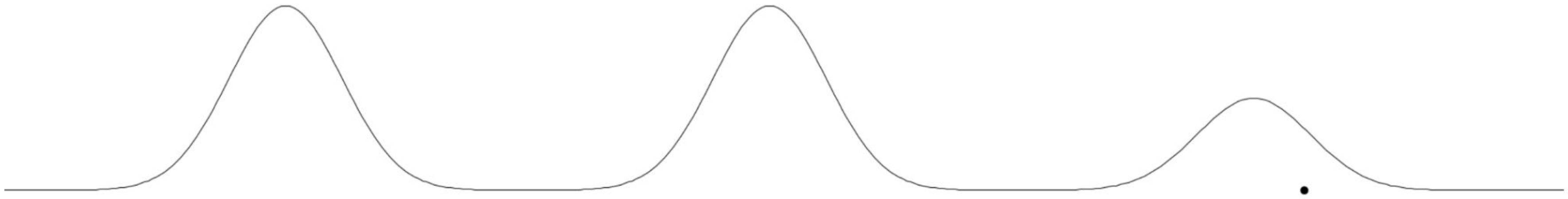
2. *Score matching:* $$\min_\theta \mathbb{E}_{p_{\text{data}}} \left[\|\nabla_x \log p(x) - s_\theta(x)\|^2\right]$$

What are the limitations of these approaches?

3. Denoising approaches. ⇒ see discussion 11 for more information.

Two related challenges for practical sample generation

1. Sampling from multimodal distributions.



2. Generating realistic samples of high-dimensional data: starting points for MCMC Langevin may be OOD, and Langevin may fail to get back to high-density areas if the score is poorly fit outside high-density areas.

Solutions?

Kernels

High-level strategy

Motivation: want to train and run a model on a high-dimensional set of features, without blowing up computational complexity

3-step process:

1. Project your features to a higher dimensional space $x \rightarrow \phi(x)$
2. Rewrite all training and inference steps using only inner products between transformed features $\phi(x_i)^T \phi(x_j)$
3. Come up with a kernel function *k* that computes these inner products between high-dimensional vectors using the raw features

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

How to figure out the appropriate kernel function?

Suppose that $x \in \mathbb{R}^d$. We want to transform *x* so that it contains all monomials with degree ≤ 3.

Roughly how big is this transformed vector to the right?

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_1^2 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_2 x_1 \\ \vdots \\ x_1^3 \\ x_1^2 x_2 \\ \vdots \end{bmatrix}.$$

How to figure out the appropriate kernel function? (cont.)

$$\begin{aligned}\langle \phi(x), \phi(z) \rangle &= 1 + \sum_{i=1}^{d} x_i z_i + \sum_{i,j \in \{1,\dots,d\}} x_i x_j z_i z_j + \sum_{i,j,k \in \{1,\dots,d\}} x_i x_j x_k z_i z_j z_k \\ &= 1 + \sum_{i=1}^{d} x_i z_i + \left(\sum_{i=1}^{d} x_i z_i \right)^2 + \left(\sum_{i=1}^{d} x_i z_i \right)^3 \\ &= 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3 \end{aligned} \tag{9}$$

Exam Tips

- Final is cumulative. Take time to review MT1 content too.
- Scope:
  - Lectures 1-27 (no special topics)
  - Homeworks 1-7
  - Discussions 0-12
- Make sure you are comfortable with probability theory, linear algebra, and matrix calculus.
  - Homework 1 is good for reviewing these concepts!
- Exam is Tuesday 12/17, 8-11am
  - Early exam, get a good night’s sleep!!
- Good luck!

Q&A