Bayesian Networks

A Bayesian network is a data structure that represents the dependencies among random variables. Bayesian networks have the following properties:

- They are directed graphs.
- Each node on the graph represent a random variable.
- An arrow from X to Y represents that X is a parent of Y. That is, the probability distribution of Y depends on the value of X.
- Each node X has probability distribution P(X | Parents(X)).
  - No parent: probability distribution for the ramdom variable P(X)
  - If parents: conditional probability distribution
  - Given ancestors of X can compute the full joint probability distribution

Bayesian Network: Example

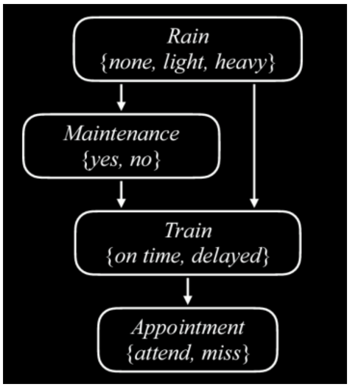
(Maintenance | Rain) - conditional distribution

Rain is the root node in this network. A random variable whose probability distribution is not reliant on any prior event.

none	light	heavy
0.7	0.2	0.1

Maintenance encodes whether there is train track maintenance, taking the values {yes, no}. **Rain** is a parent node of Maintenance, which means that the probability distribution of Maintenance is affected by Rain.

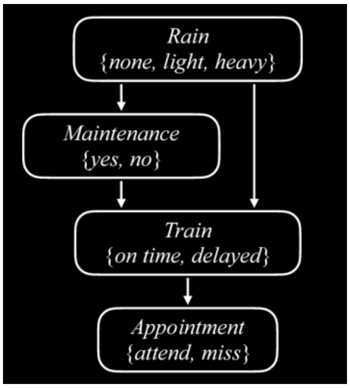
Rain	yes	no
none	0.4	0.6
light	0.2	0.8
heavy	0.1	0.9



Bayesian Network: Example
(Train | Rain, Maintenance) - conditional distribution

Train is the variable that encodes whether the train is on time or delayed, taking the values {*on time, delayed*}. **Maintenance** and **Rain** are both parents of Train, and their values affect the probability distribution of Train.

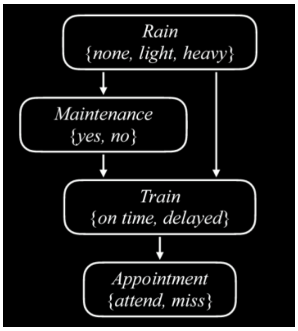
Rain	Main	Yes	No
none	yes	0.8	0.2
none	no	0.9	0.1
light	yes	0.6	0.4
light	no	0.7	0.3
heavy	yes	0.4	0.6
heavy	no	0.5	0.5



Bayesian Network: Example
(Appointment | Train) - conditional distribution

Appointment is a random variable that represents whether we attend our appointment, taking the values {*attend, miss*}. Note that its only parent is Train. This point about Bayesian network is noteworthy: parents include only direct relations. It is true that maintenance affects whether the train is on time, and whether the train is on time affects whether we attend the appointment. But what directly affects our chances of attending the appointment is whether the train came on time, and this is what is represented in the Bayesian network.

Train	Attend	Miss
on-time	0.9	0.1
delayed	0.6	0.4



Bayesian Network: Example
computing the full joint distribution

Rain

none	light	heavy
0.7	**0.2**	0.1

Maintenance

Rain	yes	no
none	0.4	0.6
light	0.2	**0.8**
heavy	0.1	0.9

Train

Rain	Main	Yes	No
none	yes	0.8	0.2
none	no	0.9	0.1
light	yes	0.6	0.4
light	no	**0.7**	0.3
heavy	yes	0.4	0.6
heavy	no	0.5	0.5

Appointment

Train	Attend	Miss
on-time	0.9	0.1
delayed	0.6	**0.4**

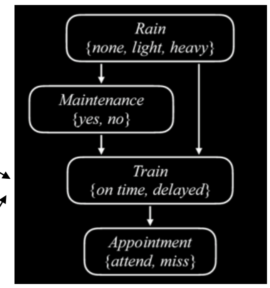
To compute **P(*light, no, delayed, miss*)**:
= P(*light*)P(*no* | *light*)P(*delayed* | *light, no*)P(*miss* | *delayed*). The value of each of the individual probabilities can be found in the probability distributions above, and then these values are multiplied to produce P(*no, light, delayed, miss*).

Inference

- Components for inferring using a Bayes' Net
  - **Query X**: variable for which to compute the probability distribution.
  - **Evidence variables E**: one or more variables that have been observed for event e.
  - **Hidden variables Y**: variables that aren't the query and haven't been observed. For example, standing at the train station, we can observe whether there is rain, but we can't know if there is maintenance on the track further down the road. Thus, Maintenance would be a hidden variable in this situation.
  - **The goal: calculate P(X | e)**. For example, compute the probability distribution of the Train variable (the query) based on the evidence e that we know there is light rain and no maintenance.
- **P**(*Appointment* | *light, no*) = αP(*Appointment, light, no*)
  = α[**P**(*Appointment, light, no, delayed*) + **P**(*Appointment, light, no, on time*)].

Computing P(Appointment | light, no)

Hidden variable: Train



$\mathbf{P}(\mathit{Appointment} \mid \mathit{light}, \mathit{no})$

$= \alpha \mathrm{P}(\mathit{Appointment}, \mathit{light}, \mathit{no})$

$= \alpha[\mathbf{P}(\mathit{Appointment}, \mathit{light}, \mathit{no}, \mathit{delayed})$
$+ \mathbf{P}(\mathit{Appointment}, \mathit{light}, \mathit{no}, \mathit{on\ time})]$.

Normalization

Inference by Enumeration (what we just did)

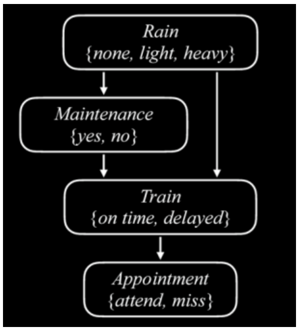
Inference by enumeration is a process of finding the probability distribution of variable X given observed evidence e and some hidden variables Y.

$$P(X|e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)$$

- X stand for the query variable,
- e for the observed evidence,
- y for all the values of the hidden variables, and
- α normalizes the result so that probabilities add up to 1

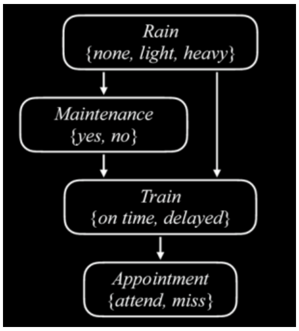
Libraries for doing probabilistic inference: E.g. Pomegranate

```
from pomegranate import *

# Rain node has no parents
rain = Node(DiscreteDistribution({
    "none": 0.7,
    "light": 0.2,
    "heavy": 0.1
}), name="rain")

# Track maintenance node is conditional on rain
maintenance = Node(ConditionalProbabilityTable([
    ["none", "yes", 0.4],
    ["none", "no", 0.6],
    ["light", "yes", 0.2],
    ["light", "no", 0.8],
    ["heavy", "yes", 0.1],
    ["heavy", "no", 0.9]
], [rain.distribution]), name="maintenance")
```



Libraries for doing probabilistic inference: E.g. Pomegranate

```
# Train node is conditional on rain and maintenance
train = Node(ConditionalProbabilityTable([
    ["none", "yes", "on time", 0.8],
    ["none", "yes", "delayed", 0.2],
    ["none", "no", "on time", 0.9],
    ["none", "no", "delayed", 0.1],
    ["light", "yes", "on time", 0.6],
    ["light", "yes", "delayed", 0.4],
    ["light", "no", "on time", 0.7],
    ["light", "no", "delayed", 0.3],
    ["heavy", "yes", "on time", 0.4],
    ["heavy", "yes", "delayed", 0.6],
    ["heavy", "no", "on time", 0.5],
    ["heavy", "no", "delayed", 0.5],
], [rain.distribution, maintenance.distribution]), name="train")

# Appointment node is conditional on train
appointment = Node(ConditionalProbabilityTable([
    ["on time", "attend", 0.9],
    ["on time", "miss", 0.1],
    ["delayed", "attend", 0.6],
    ["delayed", "miss", 0.4]
], [train.distribution]), name="appointment")
```



Libraries for doing probabilistic inference: E.g. Pomegranate

```
# Create a Bayesian Network and add states
model = BayesianNetwork()
model.add_states(rain, maintenance, train, appointment)

# Add edges connecting nodes
model.add_edge(rain, maintenance)
model.add_edge(rain, train)
model.add_edge(maintenance, train)
model.add_edge(train, appointment)

# Finalize model
model.bake()
```

```
# Print predictions for each node
for node, prediction in zip(model.states, predictions):
  if isinstance(prediction, str):
    print(f"{node.name}: {prediction}")
  else:
    print(f"{node.name}")
    for value, probability in prediction.parameters[0].items():
      print(f"    {value}: {probability:.4f}")
```

```
# Calculate probability for a given observation
probability = model.probability([["none", "no", "on time", "attend"]])
print(probability)
```

```
# Calculate predictions based on the evidence that the train was delaye
predictions = model.predict_proba({
  "train": "delayed"
})
```


Pomegranate: joint probability

```
from model import model

# Calculate probability for a given observation
probability = model.probability([["none", "no", "on time", "attend"]])
probability = model.probability([["none", "no", "on time", “miss"]])

print(probability)
```

```
workspace@Brian-MBP bayesnet % python likelihood.py
0.34019999999999995
workspace@Brian-MBP bayesnet % python likelihood.py
0.037800000000000014
workspace@Brian-MBP bayesnet %
```

Pomegranate Inference - 1
inference.py

```
from model import model

# Calculate predictions
predictions = model.predict_proba({
    "train": "delayed"
})

# Print predictions for each node
for node, prediction in zip(model.states, predictions):
    if isinstance(prediction, str):
        print(f"{node.name}: {prediction}")
    else:
        print(f"{node.name}")
        for value, probability in prediction.parameters[0].items():
            print(f"    {value}: {probability:.4f}")
```

evidence

```
rain
    none: 0.4583
    light: 0.3069
    heavy: 0.2348
maintenance
    yes: 0.3568
    no: 0.6432
train: delayed
appointment
    miss: 0.4000
    attend: 0.6000
```


Pomegranate Inference - 2
inference.py

```
rain
    none: 0.4583
    light: 0.3069
    heavy: 0.2348
maintenance
    yes: 0.3568
    no: 0.6432
train: delayed
appointment
    miss: 0.4000
    attend: 0.6000
```

```
from model import model

# Calculate predictions
predictions = model.predict_proba({
    “rain”: “heavy”,
    "train": "delayed“
})

# Print predictions for each node
for node, prediction in zip(model.states, predictions):
    if isinstance(prediction, str):
        print(f"{node.name}: {prediction}")
    else:
        print(f"{node.name}")
        for value, probability in prediction.parameters[0].items():
            print(f"    {value}: {probability:.4f}")
```

new evidence

change

no change

```
rain: heavy
maintenance
    yes: 0.1176
    no: 0.8824
train: delayed
appointment
    attend: 0.6000
    miss: 0.4000
```

Sampling: A technique for approximate inference

- Inference by enumeration as a way of computing probability can be inefficient
- Especially when there are many variables in the model.
- A different way to go about this would be abandoning exact inference in favor of approximate inference. Doing this, we lose some precision in the generated probabilities, but often this imprecision is negligible.
- By doing this we gain a scalable method of calculating probabilities
  - See CS50 notes for an example

Markov Models: Incorporating Time

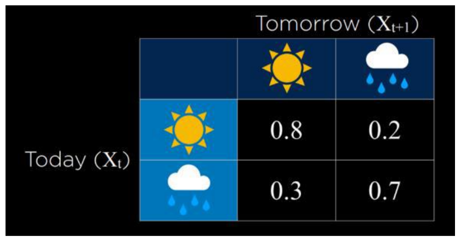
- To represent the variable of time we will create a new variable, X, and change it based on the event of interest, such that X_t is the current event, X_{t+1} is the next event, and so on. To be able to predict events in the future, we will use Markov Models.

- The **Markov** assumption is an assumption that the current state depends on only a finite fixed number of previous states.

- A **Markov chain** is a sequence of random variables where the distribution of each variable follows the Markov assumption. That is, each event in the chain occurs based on the probability of the event before it.

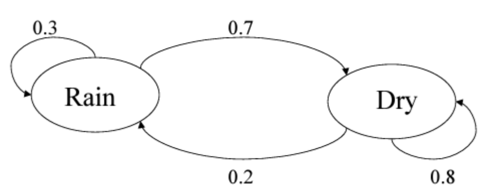
Constructing a Markov Chain
Using the transition model

- Transition Model: specifies the probability distributions of the next event based on the possible values of the current event.



Markov Model: Detailed Example

- Example Markov Model:



- Two states : 'Rain' and 'Dry'.
- Transition probabilities:
  - P('Rain'|'Rain')=0.3 , P('Dry'|'Rain')=0.7 ,
  - P('Rain'|'Dry')=0.2, P('Dry'|'Dry')=0.8
- Initial probabilities: say P('Rain')=0.4 , P('Dry')=0.6 .

Constructing a Markov Chain

- Using a transition model and a starting distribution, a sample a Markov chain can be generated.
- X_0 is either rainy or sunny
- Then sample the next day based on the probability of it being sunny or rainy given the weather today.
- Then, condition the probability of the day after tomorrow based on tomorrow, and so on, resulting in a Markov chain:

Calculation of sequence probability

- By Markov chain property, probability of state sequence can be found by the formula:

$$P(s_{i1}, s_{i2}, \ldots, s_{ik}) = P(s_{ik} \mid s_{i1}, s_{i2}, \ldots, s_{ik-1}) P(s_{i1}, s_{i2}, \ldots, s_{ik-1})$$
$$= P(s_{ik} \mid s_{ik-1}) P(s_{i1}, s_{i2}, \ldots, s_{ik-1}) = \ldots$$
$$= P(s_{ik} \mid s_{ik-1}) P(s_{ik-1} \mid s_{ik-2}) \ldots P(s_{i2} \mid s_{i1}) P(s_{i1})$$

- Suppose we want to calculate a probability of a sequence of states in our example, {'Dry','Dry','Rain',Rain'}

P({'Dry','Dry','Rain',Rain'})
= P('Rain'|'Rain') P('Rain'|'Dry') P('Dry'|'Dry') P('Dry')
= 0.3*0.2*0.8*0.6

Pomogranate

```
from pomegranate import *
# Define starting probabilities
start = DiscreteDistribution({
"sun": 0.5,
"rain": 0.5
})
```

```
# Define transition model
transitions =
ConditionalProbabilityTable([
["sun", "sun", 0.8],
["sun", "rain", 0.2],
["rain", "sun", 0.3],
["rain", "rain", 0.7]
], [start])
# Create Markov chain
model = MarkovChain([start,
transitions])
# Sample 50 states from chain
print(model.sample(50))
```


Hidden Markov Models

- A hidden Markov model is a type of a Markov model for a system where we have some observed sequence of events are the result of a sequence of hidden states.
  - E.g., an agent has some measurement of the world but no access to the precise state of the world.
  - The state of the world is called the **hidden state** and whatever data the agent has access to are the **observations**
- Examples:
  - In speech recognition, the hidden state is the words that were spoken, and the observation is the audio waveforms
  - In measuring user engagement on websites, the hidden state is how engaged the user is, and the observation is the website or app analytics for the user.

Sensors Models: Observation

Hidden State	Observation
robot's position	robot's sensor data
words spoken	audio waveforms
user engagement	website or app analytics
weather	umbrella

Example:
Estimating the weather from observation

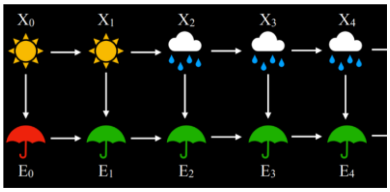
Here is our **sensor model** (or **emission** model) that represents these probabilities:

		Observation (E_t)	
			
State (X_t)		0.2	0.8
		0.9	0.1

- Assumes that the evidence variable depends only on the corresponding state.

Representation of HMM's

- HMM is represented by a Markov chain with two layers



X_i : Hidden state

E_i : Evidence

- Given earlier observations, calculate probability distribution of future state
- Given observations from 0..n compute probability distribution for past X_i
- Given observations from start until now, calculate most likely sequence of states e.g. Speech recognition: infer most likely sequence of words

Hidden Markov model

- Markov chain property: probability of each subsequent state depends only on what was the previous state:

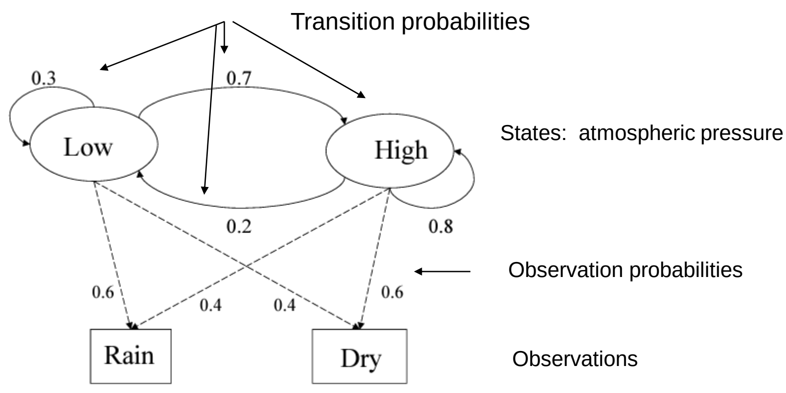
$$P(s_{ik} \mid s_{i1}, s_{i2}, \ldots, s_{ik-1}) = P(s_{ik} \mid s_{ik-1})$$

- States are not visible, but each state randomly generates one of M observations (or visible states) $\{v_1, v_2, \ldots, v_M\}$
- To define hidden Markov model, the following probabilities have to be specified:
  - matrix of transition probabilities $A=(a_{ij})$, $a_{ij} = P(s_i \mid s_j)$,
  - matrix of observation probabilities $B=(b_i(v_m))$, $b_i(v_m)=P(v_m \mid s_i)$,
  - vector of initial probabilities $\pi=(\pi_i)$,) $\pi_i = P(s_i)$.
  - Model is represented by $M=(A, B, \pi)$.

Different Example of Hidden Markov Model

- Two states : ‘Low’ and ‘High’ atmospheric pressure.
- Two observations : ‘Rain’ and ‘Dry’.
- Transition probabilities:
  - P(‘Low’|‘Low’)=0.3 , P(‘High’|‘Low’)=0.7 ,
  - P(‘Low’|‘High’)=0.2, P(‘High’|‘High’)=0.8
- Observation probabilities : P(‘Rain’|‘Low’)=0.6 , P(‘Dry’|‘Low’)=0.4 , P(‘Rain’|‘High’)=0.4 , P(‘Dry’|‘High’)=0.3 .
- Initial probabilities: say P(‘Low’)=0.4 , P(‘High’)=0.6 .

Example: Graphical Representation



Calculation of observation sequence probability

- Suppose we want to calculate a probability of a sequence of observations in our example, {‘Dry’,’Rain’}.
- Use all possible states that could result in {‘Dry’,’Rain’}.
- P({‘Dry’,’Rain’}) =
  - P({‘Dry’,’Rain’} , {‘Low’,’Low’}) + P({‘Dry’,’Rain’} , {‘Low’,’High’}) + P({‘Dry’,’Rain’} , {‘High’,’Low’}) + P({‘Dry’,’Rain’} , {‘High’,’High’})
  - Where the first term is:
    P({‘Dry’,’Rain’} , {‘Low’,’Low’})
    = P({‘Dry’,’Rain’} | {‘Low’,’Low’}) P({‘Low’,’Low’})
    = P(‘Dry’|’Low’)P(‘Rain’|’Low’) P(‘Low’)P(‘Low’|’Low)
    = 0.4*0.4*0.6*0.4*0.3

Using HMMs

- Evaluation: Given an HMM model M and an observation sequence O, calculate the probability that M generated the observation sequence O.
- Decoding: Given an HMM model M and an observation sequence O, calculate the probability that the most likely sequence of hidden states S_i that produced this observation sequence O.
- Learning: Given some training observation sequences O_1, O_2, …, O_k , and the general structure of an HMM, determine the parameters (e.g. transition probabilities, observations probabilities, …) that best fit the training data

Pomegranate

```
from pomegranate import *
# Observation model for each state
sun = DiscreteDistribution({
   "umbrella": 0.2,
   "no umbrella": 0.8
})
rain = DiscreteDistribution({
   "umbrella": 0.9,
   "no umbrella": 0.1
})
states = [sun, rain]
# Transition model
transitions = numpy.array(
   [[0.8, 0.2], # Tomorrow's predictions if today = sun
    [0.3, 0.7]] # Tomorrow's predictions if today = rain
)
```

```
# Starting probabilities
starts = numpy.array([0.5, 0.5])
# Create the model
model = HiddenMarkovModel.from_matrix(
   transitions, states, starts,
   state_names=["sun", "rain"]
)
model.bake()
```