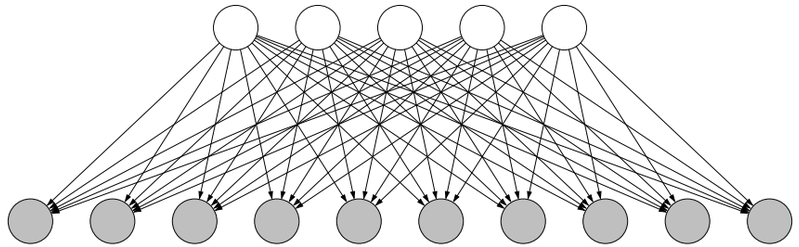
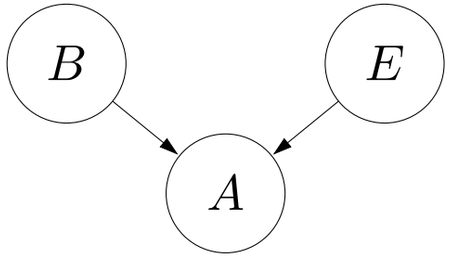
Bayesian networks: probabilistic programming



- In this module, I will talk about probabilistic programming, a new way to think about defining Bayesian networks through the lens of writing programs, which really highlights the generative process aspect of Bayesian networks.

Probabilistic programs



Joint distribution:

$$\mathbb{P}(B = b, E = e, A = a) = p(b)p(e)p(a \mid b, e)$$

Probabilistic program: alarm

$B \sim \text{Bernoulli}(\epsilon)$

$E \sim \text{Bernoulli}(\epsilon)$

$A = B \vee E$

```
def Bernoulli(epsilon):
    return random.random() < epsilon
```

Key idea: probabilistic program

A randomized program that sets the random variables.

- Recall that a Bayesian network is given by (i) a set of random variables, (ii) directed edges between those variables capturing qualitative dependencies, (iii) local conditional distributions of each variable given its parents which captures these dependencies quantitatively, and (iv) a joint distribution which is produced by multiplying all the local conditional distributions together. Now the joint distribution is your probabilistic database, which you can answer all sorts of questions on it using probabilistic inference.
- There is another way of writing down Bayesian networks other than graphically or mathematically, and that is as a probabilistic program.
- Let's go through the alarm example. We can sample B and E independently from a Bernoulli distribution with parameter ϵ, which produces 1 (true) with probability ϵ. Then we just set $A = B \vee E$.
- In general, a **probabilistic program** is a randomized program that invokes a random number generator. Executing this program will assign values to a collection of random variables $X_1, \ldots, X_n$; that is, generating an assignment.
- We then define probability under the joint distribution of an assignment to be exactly the probability that the program generates an assignment.
- While you can run the probabilistic program to generate samples, it's important to think about it as a mathematical construct that is used to define a joint distribution.

Probabilistic program: example

Probabilistic program: object tracking

$X_0 = (0, 0)$

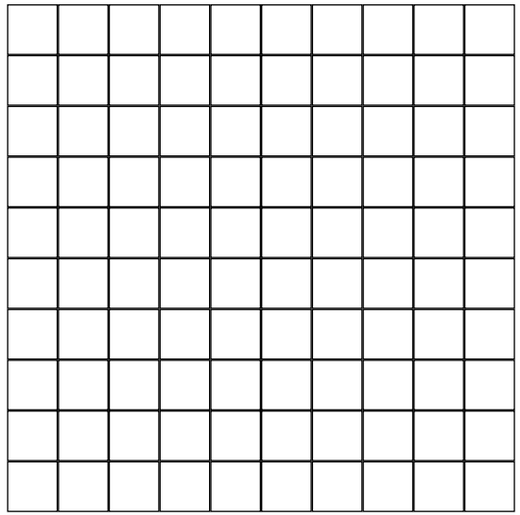
For each time step $i = 1, \ldots, n$:

    if Bernoulli(α):

        $X_i = X_{i-1} + (1, 0)$ [go right]

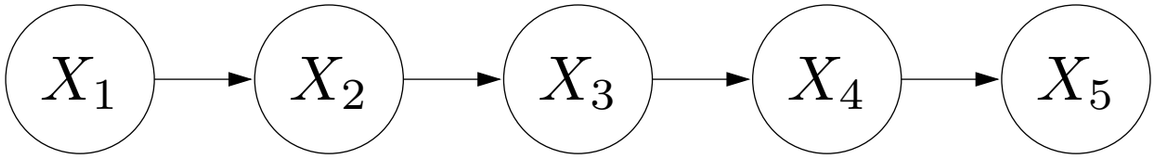
    else:

        $X_i = X_{i-1} + (0, 1)$ [go down]



(press ctrl-enter to save)

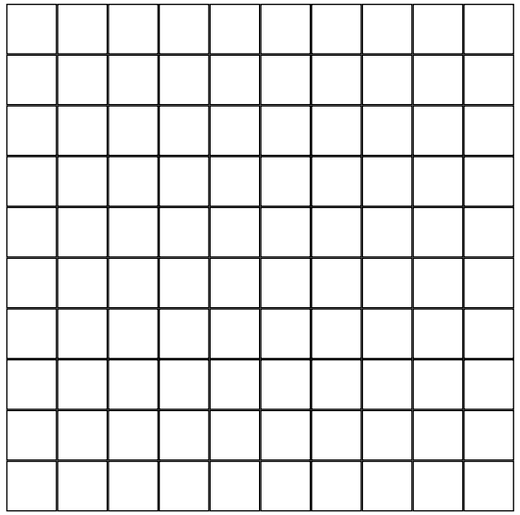
Run



- This is a more interesting example showcasing the convenience of probabilistic **programming**.
- In this program, we'll use a for loop, which allows us to compactly specify the distribution over an unboundedly large (n) set of variables.
- In the object tracking example, we define a program that generates the trajectory of an object. At each time step i, we take the previous X_{i-1} location and move it right with probability α and down with probability $1-\alpha$, yielding X_i.
- This program is a full specification of the local conditional distribution and thus the joint distribution!
- Try clicking [Run] to run the program. Each time a new assignment of $(X_1, \ldots, X_n)$ is chosen, and recall that the probability of the program generating an assignment is the probability under the joint distribution by definition.
- We can also draw the Bayesian network, which allows us to visualize the dependencies. Here, each X_i only depends on X_{i-1}. This chain-structured Bayesian network is called a **Markov model**. However, note that the graphical representation doesn't specify the local conditional distributions.

Probabilistic inference: example

Question: what are possible trajectories given **evidence** $X_{10} = (8, 2)$?



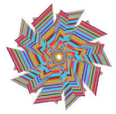
(press ctrl-enter to save)

Run

- Having used the program to define a joint distribution, we can now answer questions about that distribution.
- For example, suppose that we observe evidence $X_{10} = (8, 2)$. What is the distribution over the other variables?
- In the demo, we condition on the evidence and observe the distribution over all trajectories, which are constrained to go through $(8, 2)$ at time step 10.

Application: language modeling

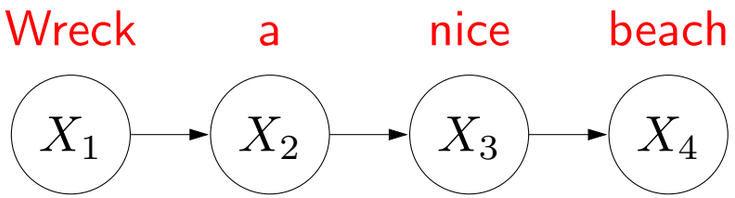
Can be used to score sentences for speech recognition or machine translation



Probabilistic program: Markov model

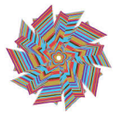
For each position $i = 1, 2, \ldots, n$:

Generate word $X_i \sim p(X_i \mid X_{i-1})$



- Now I'm going to quickly go through a set of examples of Bayesian networks or probabilistic programs and talk about the applications they are used for.
- A natural language sentence can be viewed as a sequence of words, and a language model assigns a probability to each sentence, which measures the "goodness" of that sentence.
- Markov models and higher-order Markov models (called n-gram models in NLP), were the dominant paradigm for language modeling before deep learning, and for a while, they outperformed neural language models since they were computationally much easier to scale up.
- While they could be used to generate text unconditionally, they were often used in the context of a speech recognition or machine translation system to score the fluency of the output.
- A Markov model generates each word given the previous word according to some local conditional distribution $p(X_i \mid X_{i-1})$ which we're not specifying right now.

Application: object tracking

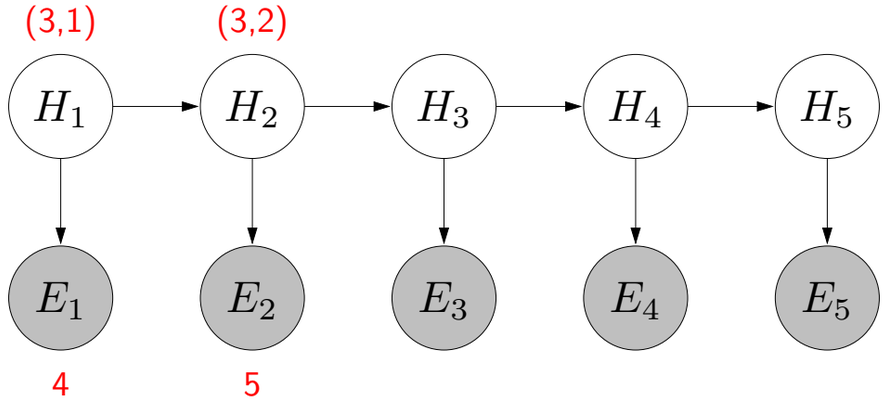


Probabilistic program: hidden Markov model (HMM)

For each time step $t = 1, \ldots, T$:

Generate object location $H_t \sim p(H_t \mid H_{t-1})$

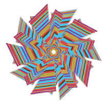
Generate sensor reading $E_t \sim p(E_t \mid H_t)$



Inference: given sensor readings, where is the object?

- Markov models are limiting because they do not have a way of talking about noisy evidence (sensor readings). They can be extended to hidden Markov models, which introduce a parallel sequence of observation variables.
- For example, in object tracking, H_t denotes the true object location, and E_t denotes the noisy sensor reading, which might be (i) the location H_t plus noise, or (ii) the distance from H_t plus noise, depending on the type of sensor.
- In speech recognition, H_t would be the phonemes or words and E_t would be the raw acoustic signal.

Application: multiple object tracking



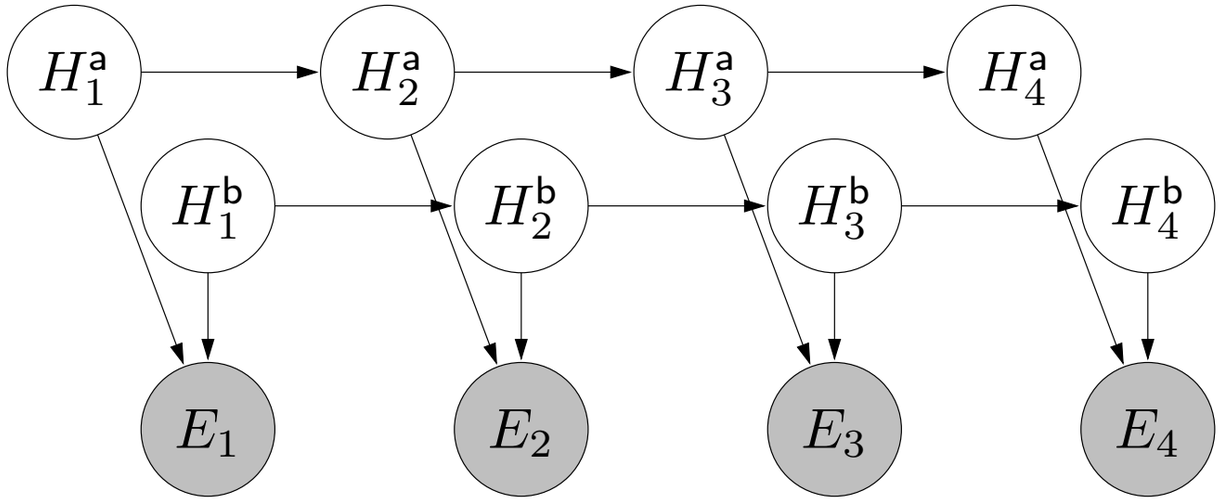
Probabilistic program: factorial HMM

For each time step $t = 1, \ldots, T$:

  For each object $o \in \{\mathsf{a}, \mathsf{b}\}$:

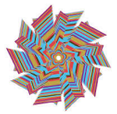
    Generate location $H_t^o \sim p(H_t^o \mid H_{t-1}^o)$

  Generate sensor reading $E_t \sim p(E_t \mid H_t^{\mathsf{a}}, H_t^{\mathsf{b}})$



- An extension of an HMM, called a **factorial HMM**, can be used to track multiple objects.
- We assume that each object moves independently according to a Markov model, but that we get one sensor reading which is some noisy aggregated function of the true positions.
- For example, E_t could be the set $\{H_t^{\mathsf{a}}, H_t^{\mathsf{b}}\}$, which reveals where the objects are, but doesn't say which object is responsible for which element in the set.

Application: document classification

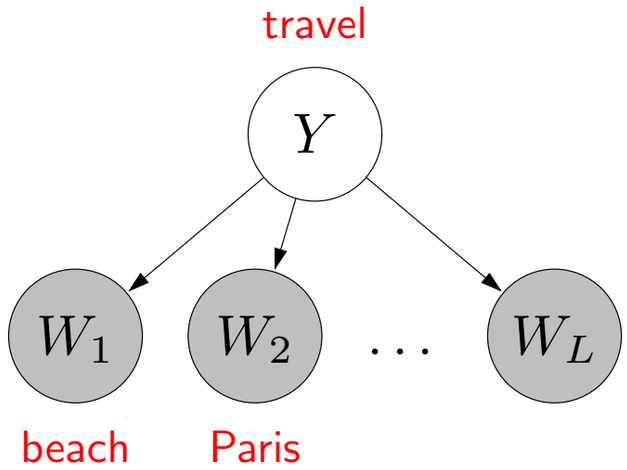


Probabilistic program: naive Bayes

Generate label $Y \sim p(Y)$

For each position $i = 1, \ldots, L$:

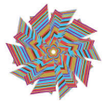
Generate word $W_i \sim p(W_i \mid Y)$



Inference: given a text document, what is it about?

- Naive Bayes is a very simple model which is often used for classification. For document classification, we generate a label and all the words in the document given that label.
- Note that the words are all generated independently, which is not a very realistic model of language, but naive Bayes models are surprisingly effective for tasks such as document classification.
- These types of models are traditionally called generative models as opposed to discriminative models for classification. Rather than thinking about how you take the input and produce the output label (e.g., using a neural network), you go the other way around: think about how the input is generated from the output (which is usually the purer, more structured form of the input).
- One advantage of using Naive Bayes for classification is that "training" is extremely easy and fast and just requires counting (as opposed to performing gradient descent).

Application: topic modeling



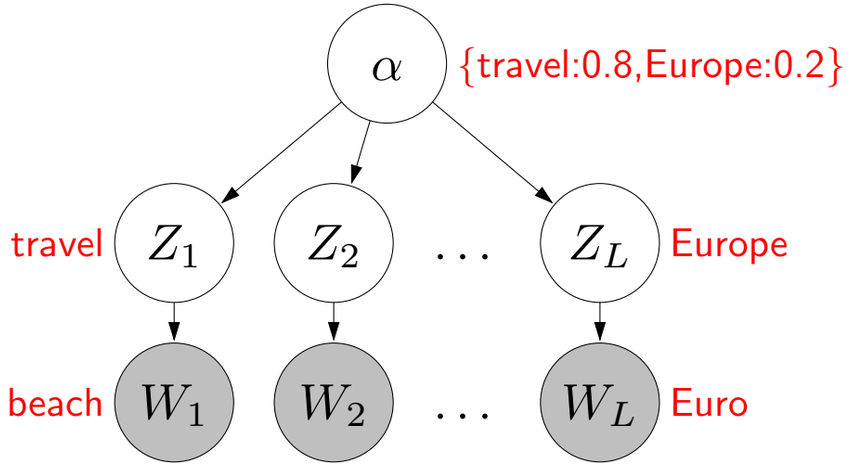
Probabilistic program: latent Dirichlet allocation

Generate a distribution over topics $\alpha \in \mathbb{R}^K$

For each position $i = 1, \ldots, L$:

Generate a topic $Z_i \sim p(Z_i \mid \alpha)$

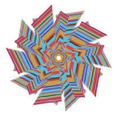
Generate a word $W_i \sim p(W_i \mid Z_i)$



Inference: given a text document, what topics is it about?

- A more sophisticated model of text is Latent Dirichlet Allocation (LDA), which allows a document to not just be about one topic or class (which was true in naive Bayes), but about multiple topics.
- Here, the distribution over topics α is chosen per document from a Dirichlet distribution. Note that α is a continuous-valued random variable. For each position, we choose a topic according to that per-document distribution and generate a word given that topic.
- Latent Dirichlet Alloction (LDA) has been very infuential for modeling not only text but images, videos, music, etc.; any sort of data with hidden structure. It is very related to matrix factorization.

Application: medical diagnosis



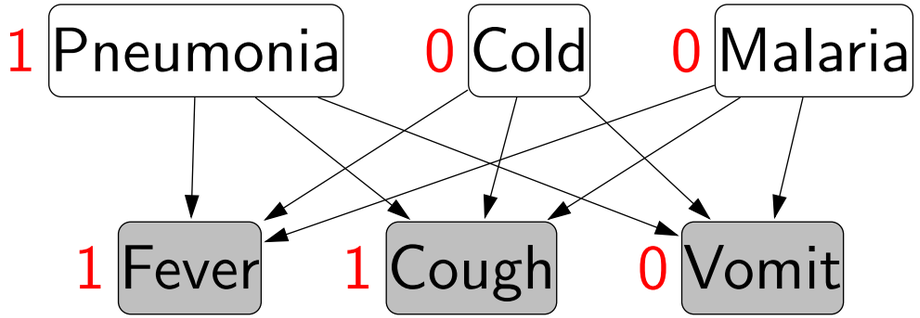
Probabilistic program: diseases and symptoms

For each disease $i = 1, \ldots, m$:

Generate activity of disease $D_i \sim p(D_i)$

For each symptom $j = 1, \ldots, n$:

Generate activity of symptom $S_j \sim p(S_j \mid D_{1:m})$



Inference: If a patient has some symptoms, what diseases do they have?

- We already saw a special case of this model. In general, we would like to diagnose many diseases and might have measured many symptoms and vitals.

Application: social network analysis

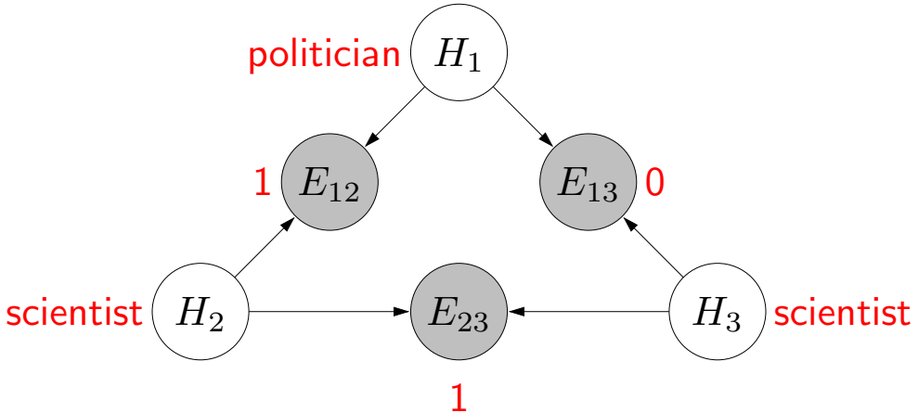
Probabilistic program: stochastic block model

For each person $i = 1, \ldots, n$:

  Generate person type $H_i \sim p(H_i)$

For each pair of people $i \neq j$:

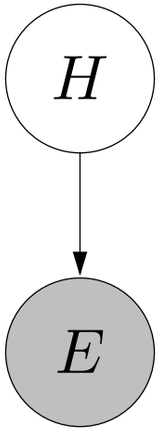
  Generate connectedness $E_{ij} \sim p(E_{ij} \mid H_i, H_j)$



Inference: Given a social network graph, what types of people are there?

- One can also model graphs such as social networks. A very naive-Bayes-like model is that each node (person) has a "type". Whether two people interact with each other (there is an edge beween the two people) is determined solely by their types and random chance.
- Note: there are extensions called mixed membership models which, like LDA, allow each person to have multiple types.

Summary



- Probabilistic program specifies a Bayesian network
- Many different types of models
- Common paradigm: come up with stories of how the quantities of interest (output) generate the data (input)
- Opposite of how we normally do classification!

- In summary, we've seen how we can define Bayesian networks (and therefore joint distributions) by writing down probabilistic programs.
- Using this powerful tool, we then did a whirlwind tour of lots of probabilistic programs that exist in the literature (though not often introduced under this general framework).
- The common theme of these probabilistic programs is that each attempts to produce **stories** of how certain quantities of interest H (e.g., actual location of an object) generate (or give rise to) observations E (e.g., usually noisy verions).
- After defining such a model, one can do probabilistic inference to compute $\mathbb{P}(H \mid E = e)$. Note that we can see how Bayesian networks allow us to handle heterogenous inputs (e.g., missing information). We can simply condition on partial evidence.
- Bayesian networks therefore provide quite a different paradigm compared to normal classification (e.g., neural networks). You have to think about going from the output to the input rather than input to output, which takes some getting used to.