Let’s break down how to build a directed graphical model (Bayesian network) for the medical domain data, focusing on the roles of the **joint distribution** (`joint.dat`), the **sample data** (`dataset.dat`), and the **graph** you design. I’ll explain how these components work together to achieve the goals of modeling probabilities, estimating parameters, measuring accuracy, and answering queries. The explanation will be clear and intuitive, avoiding unnecessary complexity.

Overview of the Problem

You want to model the relationships between 12 binary variables (e.g., `IsSummer` , `HasFlu` , `Coughs` , etc.) to help doctors diagnose diseases based on symptoms and other factors. The three key components are:

1. **Joint Distribution** (`joint.dat`): The "true" probability of every possible combination of the 12 variables.
2. **Sample Data** (`dataset.dat`): Observed examples (samples) from the joint distribution.
3. **Graph**: A directed acyclic graph (DAG) you design to represent dependencies (e.g., `HasFlu` causes `Coughs`).

These components work together to:

- **Model** the probability distribution efficiently.
- **Learn** parameters from data.
- **Evaluate** how close your model is to the truth.
- **Answer** queries like "What’s the probability of flu given a fever?"

Step-by-Step Explanation

1. The Joint Distribution (`joint.dat`)

What is it?

- The joint distribution lists the probability of every possible combination of the 12 binary variables (True/False).
- Since each variable is binary, there are $2^{12} = 4096$ possible combinations (e.g., all False, only `IsSummer` =True, etc.).
- Each combination is encoded as an integer from 0 to 4095:
  - Convert the 12-bit binary string (e.g., `000000000001` for `IsSummer` =True, others False) to a decimal number.
  - Example: `000000000001` = 1, `000000000010` = 2, etc.
- `joint.dat` has 4096 lines, each with a pair: `Integer Probability`.
  - Example: `0 0.0001` means the probability of all variables being False is 0.0001.

Role:

- It's the **ground truth**. It tells you the exact probability of any combination of variables.
- You use it to:
  - Compare your model's probabilities to the true ones (for accuracy).
  - Compute true probabilities for queries to validate your model's answers.

Why is it impractical alone?

- Storing and computing with 4096 probabilities is inefficient.
- It doesn't reveal dependencies (e.g., `HasFlu` causing `Coughs`).
- A graphical model simplifies this by assuming dependencies, reducing the number of parameters needed.

---

2. The Graph (Your Bayesian Network)

What is it?

- The graph is a **directed acyclic graph (DAG)** you design to represent how variables depend on each other.
- Nodes are the 12 variables.
- Edges show dependencies (e.g., an edge from `HasFlu` to `Coughs` means flu causes coughing).
- Example structure (based on medical intuition):
  - `IsSummer` → `HasFlu`, `HasFoodPoisoning`, `HasHayFever`, `HasPneumonia` (season affects disease likelihood).
  - `HasFlu` → `Coughs`, `IsFatigued`, `HasFever` (flu causes these symptoms).
  - Similarly for other diseases and symptoms.

How to design it?

- Use domain knowledge:
  - **Season**: Summer increases food poisoning risk, decreases flu.
  - **Diseases**: Flu, pneumonia, etc., cause specific symptoms.
  - **Symptoms**: Coughing can result from flu, pneumonia, or hay fever.
- Example DAG:

```text
IsSummer ⟶ HasFlu ⟶ Coughs, IsFatigued, HasFever
         ⟶ HasFoodPoisoning ⟶ Vomits, HasGastricProblems, HasFever
         ⟶ HasHayFever ⟶ HasRespiratoryProblems, HasRash, Coughs
         ⟶ HasPneumonia ⟶ HasRespiratoryProblems, Coughs, IsFatigued, HasFe
```

- Each node has a **conditional probability table (CPT)**:
  - For a node X with parents $\mathrm{Pa}(X)$, the CPT gives $P(X \mid \mathrm{Pa}(X))$.
  - Example: For `Coughs` with parents `HasFlu`, `HasHayFever`, `HasPneumonia`, the CPT has $2^3 = 8$ entries (one for each parent combination).

Role:

- The graph **simplifies the joint distribution** using the chain rule:

$$P(X_1, X_2, \ldots, X_{12}) = \prod_{i=1}^{12} P(X_i \mid \text{Pa}(X_i))$$

- Instead of 4096 parameters, you only need parameters for each CPT (e.g., 82 total in the example graph).
- It encodes **independencies** (e.g., `Coughs` is independent of `Vomits` given diseases).

How it helps:

- Reduces storage and computation.
- Makes it easier to learn probabilities from data.
- Enables efficient query answering by leveraging the structure.

---

3. The Sample Data (`dataset.dat`)

What is it?

- `dataset.dat` contains samples (observed cases) drawn from the joint distribution.
- Each line is an integer (0 to 4095) representing a complete assignment to the 12 variables.
- Example: If a line is `3` , it means `IsSummer` =True, `HasFlu` =True (binary: `000000000011`).

Role:

- You use the samples to **estimate the CPTs** of your graph.
- **How**:
  - For each node X_i and each parent configuration $\text{Pa}(X_i)$:
    - Count how often $X_i = \text{True}$ and $X_i = \text{False}$ for that parent configuration in the samples.
    - Normalize to get probabilities: $P(X_i = \text{True} \mid \text{Pa}(X_i)) = \frac{\text{Count}(X_i=\text{True}, \text{Pa}(X_i))}{\text{Count}(\text{Pa}(X_i))}$.
  - Example: For `Coughs` with parents `HasFlu`, `HasHayFever`:
    - Count cases where `HasFlu` =True, `HasHayFever` =False, and `Coughs` =True.
    - Divide by total cases where `HasFlu` =True, `HasHayFever` =False.
- If a parent configuration has no samples, use smoothing (e.g., assume 0.5 probability).

Why important?

- The samples let you learn the model's parameters from real data, making it match the true distribution as closely as possible.
- More samples → better estimates.

How They Work Together

Here's how the joint distribution, graph, and sample data interact to achieve your goals:

1. **Building the Model**:
   - **Graph**: You design the DAG based on intuition (e.g., diseases cause symptoms).
   - **Sample Data**: Use `dataset.dat` to estimate CPTs for each node in the graph.
   - **Result**: A Bayesian network that approximates the joint distribution with fewer parameters.

2 **Computing Probabilities**:

- For any assignment (e.g., `IsSummer` =True, `HasFlu` =True, `Coughs` =True, etc.):
  - Use the graph's CPTs and the chain rule to compute the probability:

$$P(\text{assignment}) = P(\text{IsSummer}) \cdot P(\text{HasFlu} \mid \text{IsSummer}) \cdot P(\text{Coughs} \mid \text{Ha}$$

  - Each term comes from the corresponding node's CPT.

3 **Measuring Accuracy**:

- **Joint Distribution**: Compare your model's probability for each of the 4096 assignments to the true probabilities in `joint.dat`.
- **L1 Distance**: Sum the absolute differences:

$$\text{L1} = \sum_{i=0}^{4095} |P_{\text{model}}(a_i) - P_{\text{true}}(a_i)|$$

- **KL-Divergence** (optional): Measures how much your model's distribution diverges from the true one:

$$\text{KL} = \sum_{i=0}^{4095} P_{\text{true}}(a_i) \log \frac{P_{\text{true}}(a_i)}{P_{\text{model}}(a_i)}$$

- Lower values mean your model is closer to the truth.

4 **Answering Queries**:

- A query specifies:
  - **Observed variables** (e.g., `HasFever` =True, `Coughs` =True).
  - **Query variables** (e.g., `HasFlu`).
- **Process**:
  - Enumerate all assignments consistent with the observed variables.
  - Compute the probability of each using the model's CPTs.
  - Sum probabilities to get the distribution over query variables (marginalizing out others).
  - Normalize to ensure probabilities sum to 1.
- **Compare to Truth**: Use `joint.dat` to compute the same query directly (sum probabilities of consistent assignments) and check if your model's answers match.

5 **Example Query**:

- Query: $P(\text{HasFlu} \mid \text{HasFever} = \text{True}, \text{Coughs} = \text{True})$.
- **Model**:
  - Loop through all 4096 assignments.
  - Keep only those where `HasFever` =True, `Coughs` =True.
  - Compute $P(\text{assignment})$ using the graph's CPTs.
  - Sum probabilities where `HasFlu` =True and `HasFlu` =False.
  - Normalize: $P(\text{HasFlu} = \text{True} \mid \ldots) = \frac{\text{Sum}_{\text{HasFlu=True}}}{\text{Total Sum}}$.
- **True Distribution**:
  - Use `joint.dat` to sum probabilities of consistent assignments.
  - Compare results to validate.

Practical Steps to Build It

1. **Design the Graph**:
   - Draw nodes for all 12 variables.
   - Add edges based on causality (e.g., `HasFlu` → `Coughs`).
   - List parents for each node (e.g., `Coughs`: `HasFlu`, `HasHayFever`, `HasPneumonia`).
2. **Set Up CPTs**:
   - For each node, create a CPT with $2^{|\text{parents}|}$ entries.
   - Initially, use random probabilities (summing to 1 for each parent configuration).
3. **Learn from Samples**:
   - Read `dataset.dat`.
   - For each node and parent configuration:
     - Count occurrences in the samples.
     - Compute probabilities (with smoothing for unseen cases).
   - Update CPTs with these probabilities.
4. **Compute Probabilities**:
   - Write a function to compute $P(\text{assignment})$ using the chain rule and CPTs.
   - Test on a few assignments (e.g., integer 0, 1, 3).
5. **Measure Accuracy**:
   - Read `joint.dat` to get true probabilities.
   - Compute your model's probability for each of the 4096 assignments.
   - Calculate L1 distance (and optionally KL-divergence).

6. **Answer Queries**:
   - Write a function to enumerate assignments, filter by observed variables, and sum probabilities for query variables.
   - Test queries like:
     - $P(\text{HasFlu} \mid \text{HasFever} = \text{True}, \text{Coughs} = \text{True})$.
     - $P(\text{Vomits} \mid \text{IsSummer} = \text{True})$.
     - Custom query: $P(\text{HasFlu} \mid \text{HasFever} = \text{True}, \text{HasFoodPoisoning} = \text{False})$.
7. **Validate**:
   - Compute the same queries using `joint.dat` (sum probabilities of consistent assignments).
   - Compare to ensure accuracy.

Why It Works

- **Joint Distribution**: Provides the true probabilities for validation.
- **Graph**: Reduces complexity by assuming dependencies, making computation and learning feasible.
- **Sample Data**: Lets you estimate probabilities that reflect the real world.
- **Together**:
  - The graph and samples build a model that approximates the joint distribution.
  - The joint distribution ensures your model is accurate and queries are correct.
  - The result is a tool for diagnosis (e.g., inferring diseases from symptoms).

Example Workflow

Suppose you want to answer: $P(\text{HasFlu} \mid \text{HasFever} = \text{True})$.

1. **Graph**: Your DAG has `HasFlu` → `HasFever`.
2. **Samples**: You estimated $P(\text{HasFever} = \text{True} \mid \text{HasFlu} = \text{True}) = 0.8$, $P(\text{HasFever} = \text{True} \mid \text{HasFlu} = \text{False}) = 0.2$, etc.
3. **Query**:
   - Enumerate assignments where `HasFever` =True.
   - Compute probabilities using CPTs.
   - Sum for `HasFlu` =True and `HasFlu` =False.
   - Normalize to get $P(\text{HasFlu} = \text{True} \mid \text{HasFever} = \text{True})$.
4. **Validation**: Use `joint.dat` to compute the same probability and compare.

Tips

- **Graph Design**: Keep it simple but realistic. Too many parents increase parameters and computation.
- **Smoothing**: Handle unseen parent configurations in `dataset.dat` by assuming uniform probabilities (e.g., 0.5).
- **Efficiency**: Since the domain is small (4096 assignments), exhaustive enumeration is fine. For larger problems, use approximate inference (e.g., sampling).
- **Queries**: Choose queries that test different reasoning types:
  - **Causal**: Disease → Symptom (e.g., pneumonia → cough).
  - **Evidential**: Symptom → Disease (e.g., fever → flu).
  - **Inter-causal**: Disease → Disease (e.g., flu vs. food poisoning given fever).

If you need help with specific code, designing the graph, or understanding a particular query, let me know! Would you like me to provide a simplified version of the code or focus on a specific query example?