Heredity

Description

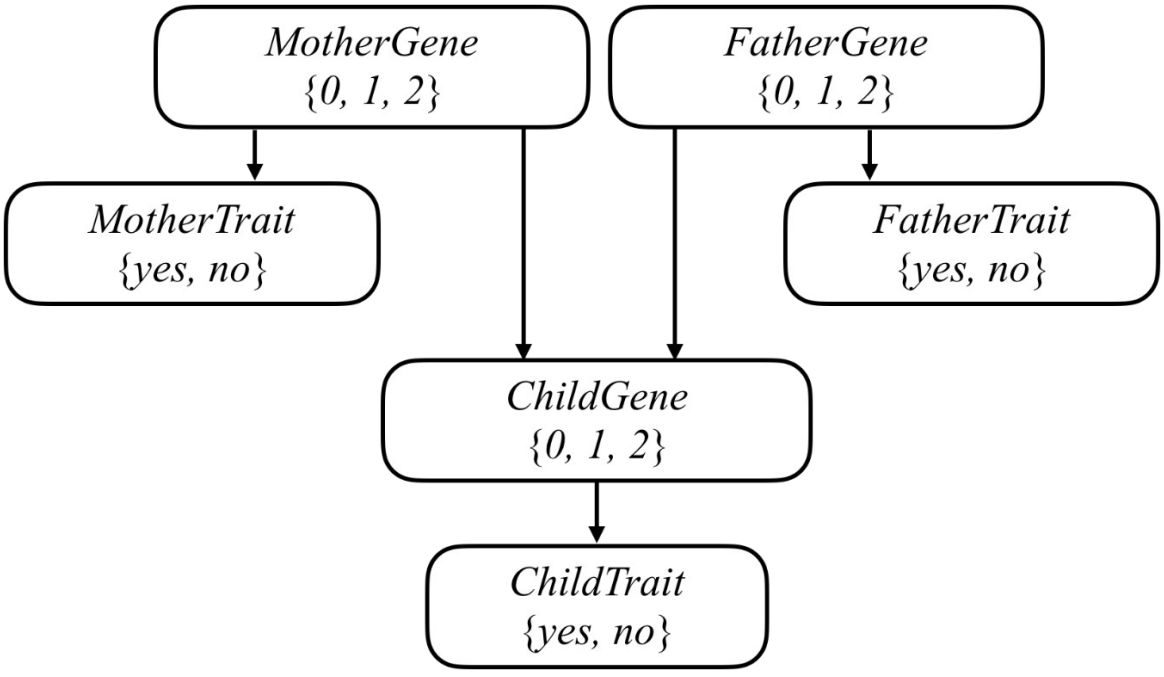
Write an AI to assess the likelihood that a person will have a particular genetic trait.

Introduction

Mutated versions of the GJB2 gene are one of the leading causes of hearing impairment in newborns. Each person carries two versions of the gene, so each person has the potential to possess either 0, 1, or 2 copies of the hearing impairment version GJB2. Unless a person undergoes genetic testing, though, it's not so easy to know how many copies of mutated GJB2 a person has. This is some "hidden state": information that has an effect that we can observe (hearing impairment), but that we don't necessarily directly know. After all, some people might have 1 or 2 copies of mutated GJB2 but not exhibit hearing impairment, while others might have no copies of mutated GJB2 yet still exhibit hearing impairment.

Every child inherits one copy of the GJB2 gene from each of their parents. If a parent has two copies of the mutated gene, then they will pass the mutated gene on to the child; if a parent has no copies of the mutated gene, then they will not pass the mutated gene on to the child; and if a parent has one copy of the mutated gene, then the gene is passed on to the child with probability 0.5. After a gene is passed on, though, it has some probability of undergoing additional mutation: changing from a version of the gene that causes hearing impairment to a version that doesn't, or vice versa.

We can attempt to model all these relationships by forming a Bayesian Network of all the relevant variables, as in the one below, which considers a family of two parents and a single child.

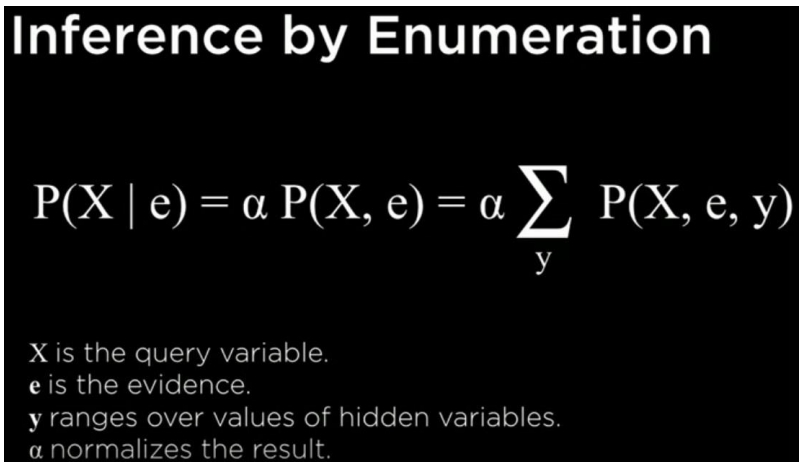


Each person in the family has a `Gene` random variable representing how many copies of a particular gene (e.g., the hearing impairment version of GJB2) a person has: a value that is 0, 1, or 2. Each person in the family also has a `Trait` random variable, which is `yes` or `no` depending on whether that person expresses a trait (e.g., hearing impairment) based on that gene. There's an arrow from each person's `Gene` variable to their `Trait` variable to encode the idea that a person's genes affect the probability that they have a particular trait. Meanwhile, there's also an arrow from both the mother and father's `Gene` random variable to their child's `Gene` random variable: the child's genes are dependent on the genes of their parents.

Your task in this project is to use this model to make inferences about a population. Given information about people, who their parents are, and whether they have a particular observable trait (e.g. hearing loss) caused by a given gene, your AI will infer the probability distribution for each person's genes, as well as the probability distribution for whether any person will exhibit the trait in question.

Mathematical Dive and Deeper Understanding of the Problem

This exercise is an example of "inference by enumeration", in which conditional probabilities could be computed for query variables given the evidence (i.e. the evidence variables) by marginalization over all the other hidden variables (i.e. by adding together all the possible enumerations of the joint probability).



The query variable is the variable for which we are trying to compute/predict the probability. In this problem, this query variable could be the probability that the child exhibits the trait. The evidence then could be the observations we make about the child's parents on whether they exhibit the trait. Finally, the hidden variables could in this case be the number of copies of genes the parents or the child have.

To draw an analogy to help understand the normalization constant α, recall the law you already know: $P(A,B)=P(A|B)*P(B)$. In that case, the joint probability was scaled/normalized by P(B).

Understanding the Environment

Look at one of the sample data sets in the `data` directory by opening `data/family0.csv` (you can open it up in a text editor, or in a spreadsheet application like Google Sheets, Excel, or Apple Numbers). Notice that the first row defines the columns for this CSV file: `name`, `mother`, `father`, and `trait`. The next row indicates that Harry has Lily as a mother, James as a father, and the empty cell for `trait` means we don't know whether Harry has the trait or not. James, meanwhile, has no parents listed in our data set (as indicated by the empty cells for `mother` and `father`), and does exhibit the trait (as indicated by the `1` in the `trait` cell). Lily, on the other hand, also has no parents listed in the data set, but does not exhibit the trait (as indicated by the `0` in the `trait` cell).

Open `heredity.py` and look first at the definition of `PROBS`. `PROBS` is a dictionary containing a number of constants representing probabilities of various events. All these events have to do with how many copies of a particular gene a person has (hereafter referred to as simply "the gene"), and whether a person exhibits a particular trait (hereafter referred to as "the trait") based on that gene. The data here is loosely based on the probabilities for the hearing impairment version of the GJB2 gene and the hearing impairment trait, but by changing these values, you could use your AI to draw inferences about other genes and traits as well!

First, `PROBS["gene"]` represents the unconditional probability distribution over the gene (i.e., the probability if we know nothing about that person's parents). Based on the data in the distribution code, in the population, there's a 1% chance of having 2 copies of the gene, a 3% chance of having 1 copy of the gene, and a 96% chance of having 0 copies of the gene.

Next, `PROBS["trait"]` represents the conditional probability that a person exhibits a trait (like hearing impairment). This is actually three different probability distributions: one for each possible value for `gene`. So `PROBS["trait"][2]` is the probability distribution that a person has the trait given that they have two versions of the gene: in this case, they have a 65% chance of exhibiting the trait, and a 35% chance of not exhibiting the trait. Meanwhile, if a person has 0 copies of the gene, they have a 1% chance of exhibiting the trait, and a 99% chance of not exhibiting the trait.

Finally, `PROBS["mutation"]` is the probability that a gene mutates from being the gene in question to not being that gene, and vice versa. If a mother has two versions of the gene, for example, and therefore passes one on to her child, there's a 1% chance it mutates into not being the target gene anymore. Conversely, if a mother has no versions of the gene, and therefore does not pass it onto her child, there's a 1% chance it mutates into being the target gene. It's therefore possible that even if neither parent has any copies of the gene in question, their child might have 1 or even 2 copies of the gene.

Ultimately, the probabilities you calculate will be based on these values in `PROBS`.

Now, look at the `main` function. The function first loads data from a file into a dictionary `people`. `people` maps each person's name to another dictionary containing information about them: including their name, their mother (if one is listed in the data set), their father (if one is listed in the data set), and whether they are observed to have the trait in question (`True` if they do, `False` if they don't, and `None` if we don't know).

Next, `main` defines a dictionary of `probabilities`, with all probabilities initially set to 0. This is ultimately what your project will compute: for each person, your AI will calculate the probability distribution over how many of copies of the gene they have, as well as whether they have the trait or not. `probabilities["Harry"]["gene"][1]`, for example, will be the probability that Harry has 1 copy of the gene, and `probabilities["Lily"]["trait "][False]` will be the probability that Lily does not exhibit the trait.

If unfamiliar, this `probabilities` dictionary is created using a Python dictionary comprehension, which in this case creates one key/value pair for each `person` in our dictionary of `people`.

Ultimately, we're looking to calculate these probabilities based on some evidence: given that we know certain people do or do not exhibit the trait, we'd like to determine these probabilities. Recall that we can calculate a conditional probability by summing up all the joint probabilities that satisfy the evidence, and then normalize those probabilities so that they each sum to 1. Your task in this project is to implement three functions to do just that: `joint_probability` to compute a joint probability, `update` to add the newly computed joint probability to the existing probability distribution, and then `normalize` to ensure all probability distributions sum to 1 at the end.

Task Specification

Complete the implementations of `joint_probability`, `update`, and `normalize`.

The `joint_probability` function should take as input a dictionary of people, along with data about who has how many copies of each of the genes, and who exhibits the trait. The function should return the joint probability of all those events taking place.

- The function accepts four values as input: `people`, `one_gene`, `two_genes`, and `have_trait`.
  - `people` is a dictionary of people as described in the "Understanding" section. The keys represent names, and the values are dictionaries that contain `mother` and `father` keys. You may assume that either `mother` and `father` are both blank (no parental information in the data set), or `mother` and `father` will both refer to other people in the `people` dictionary.
  - `one_gene` is a set of all people for whom we want to compute the probability that they have one copy of the gene.
  - `two_genes` is a set of all people for whom we want to compute the probability that they have two copies of the gene.
  - `have_trait` is a set of all people for whom we want to compute the probability that they have the trait.
  - For any person not in `one_gene` or `two_genes`, we would like to calculate the probability that they have no copies of the gene; and for anyone not in `have_trait`, we would like to calculate the probability that they do not have the trait.
- For example, if the family consists of Harry, James, and Lily, then calling this function where `one_gene = {"Harry"}`, `two_genes ={"James"}`, and `have_trait = {"Harry", "James"}` should calculate the probability that Lily has zero copies of the gene, Harry has one copy of the gene, James has two copies of the gene, Harry exhibits the trait, James exhibits the trait, and Lily does not exhibit the trait.

- For anyone with no parents listed in the data set, use the probability distribution `PROBS["gene"]` to determine the probability that they have a particular number of the gene.
- For anyone with parents in the data set, each parent will pass one of their two genes on to their child randomly, and there is a `PROBS["mutation"]` chance that it mutates (goes from being the gene to not being the gene, or vice versa).
- Use the probability distribution `PROBS["trait"]` to compute the probability that a person does or does not have a particular trait.

The `update` function adds a new joint distribution probability to the existing probability distributions in `probabilities`.

- The function accepts five values as input: `probabilities`, `one_gene`, `two_genes`, `have_trait`, and `p`.
  - `probabilities` is a dictionary of people as already described. Each person is mapped to a `"gene"` distribution and a `"trait"` distribution.
  - `one_gene` is a set of people with one copy of the gene in the current joint distribution.
  - `two_genes` is a set of people with two copies of the gene in the current joint distribution.
  - `have_trait` is a set of people with the trait in the current joint distribution.
  - `p` is the probability value of the joint distribution.
- For each `person` in `probabilities`, the function should update the `probabilities[person]["gene"]` distribution and `probabilities[person]["trait"]` distribution by adding `p` to the appropriate value in each distribution. All other values should be left unchanged.
- For example, if `"Harry"` were in both `two_genes` and in `have_trait`, then `p` would be added to `probabilities["Harry"]["gene"][2]` and to `probabilities["Harry"]["trait"][True]`.
- The function should not return any value: it just needs to update the `probabilities` dictionary.

The `normalize` function updates a dictionary of probabilities such that each probability distribution is normalized (i.e., sums to 1, with relative proportions the same).

- The function accepts a single value: `probabilities`.
  - `probabilities` is a dictionary of people as described in the "Understanding" section. Each person is mapped to a `"gene"` distribution and a `"trait"` distribution.
- For both of the distributions for each person in `probabilities`, this function should normalize that distribution so that the values in the distribution sum to 1, and the relative values in the distribution are the same.
- For example, if `probabilities["Harry"]["trait"][True]` were equal to `0.1` and `probabilities["Harry"]["trait"][False]` were equal to `0.3`, then your function should update the former value to be `0.25` and the latter value to be `0.75`: the numbers now sum to `1`, and the latter value is still three times larger than the former value.
- The function should not return any value: it just needs to update the `probabilities` dictionary.

Example Joint Probability

To help you think about how to calculate joint probabilities, we've included below an example.

Consider the following value for `people`:

```
{
    'Harry': {'name': 'Harry', 'mother': 'Lily', 'father': 'James', 'trait': None},
    'James': {'name': 'James', 'mother': None, 'father': None, 'trait': True},
    'Lily': {'name': 'Lily', 'mother': None, 'father': None, 'trait': False}
}
```

We will here show the calculation of `joint_probability(people,{"Harry"},{"James"},{"James"})`.

Based on the arguments, `one_gene` is `{"Harry"}`, `two_genes` is `{"James"}`, and `have_trait` is `{"James"}`. We will calculate the joint probability that comprises all the persons listed in `people`: Harry, James and Lily. If the person is not listed in `one_gene` or `two_genes`, we calculate the probability that they have 0 copies of the gene; and for anyone not in `have_trait`, we calculate the probability that they do not have the trait.

Therefore, we want to calculate the probability that: Lily has 0 copies of the gene and does not have the trait, Harry has 1 copy of the gene and does not have the trait, and James has 2 copies of the gene and does have the trait. You should notice that person's trait status for the current joint probability is specified in `have_trait`. If the person is on the `have_trait`, then the value of the 'trait' is True; and if the person is not on the list, then the value of the 'trait' is False. The `people` shows the values of the 'trait', but that information is compared in the main function before assigning the persons to the list of `have_trait`. Thus, you don't need to consider those 'trait' values presented in `people`, only information specified by being (or not being) in the list of `have_trait`.

We start with Lily (the order that we consider people does not matter, so long as we multiply the correct values together, since multiplication is commutative). Lily has 0 copies of the gene with probability `0.96` (this is `PROBS["gene"][0]`). Given that she has 0 copies of the gene, she doesn't have the trait with probability `0.99` (this is `PROBS["trait"][0][False]`). Thus, the probability that she has 0 copies of the gene and she doesn't have the trait is `0.96 * 0.99 = 0.9504`.

Next, we consider James. James has 2 copies of the gene with probability `0.01` (this is `PROBS["gene"][2]`). Given that he has 2 copies of the gene, the probability that he does have the trait is `0.65`. Thus, the probability that he has 2 copies of the gene and he does have the trait is `0.01 * 0.65 = 0.0065`.

Both Lily and James are parents, as they have no parents in the `people`. Harry is a child, as he has parents in the people. Each parent will pass one of their two genes on to their child randomly, and there is a `PROBS["mutation"]` chance that it mutates (goes from being the gene to not being the gene, or vice versa). You should notice that the probabilities for parents and children are calculated differently.

What's the probability that Harry has 1 copy of the gene? There are two ways this can happen. Either he gets the gene from his mother and not his father, or he gets the gene from his father and not his mother. His mother Lily has 0 copies of the gene, so Harry will get the gene from his mother with probability `0.01` (this is `PROBS["mutation"]`), since the only way to get the gene from his mother is if it mutated; conversely, Harry will not get the gene from his mother with probability `0.99`. His father James has 2 copies of the gene, so Harry will get the gene from his father with probability `0.99` (this is `1 - PROBS["mutation"]`), and will not get the gene from his father with probability `0.01` (the chance of a mutation). So, the probability that Harry will get 1 copy from father and not any from mother is `0.99 * 0.99`, and the probability that Harry will get 0 copy from father and 1 copy from mother is `0.01 * 0.01`. Both of these cases can be added together to get `0.99 * 0.99 + 0.01 * 0.01 = 0.9802`, the probability that Harry has 1 copy of the gene.

Given that Harry has 1 copy of the gene, the probability that he does not have the trait is `0.44` (this is `PROBS["trait"][1][False]`). So, the probability that Harry has 1 copy of the gene and does not have the trait is `0.9802 * 0.44 = 0.431288`.

Therefore, the entire joint probability is just the result of multiplying all these values for each of the three people: `0.9504 * 0.0065 * 0.431288 = 0.0026643247488`.

Hints for generalizing the calculation of the `joint_probability` function for other cases: In case of Harry, we have shown above the probabilities, where a parent has either 0 (Lily) or 2 (James) copies of the gene. If a parent has 1 copy of the gene, then the probability to give 1 copy of the gene to his/her child is `0.5`. Harry, a child, was considered to have 1 copy of the gene, but you must also think about that what would be the probabilities if a child has 0 or 2 copies of the genes. In case of a child having 2 copies of the gene, both parents give one copy of the gene; and in case of a child having 0 copies of the gene, both parents give 0 copy of the gene. For instance, if Lily has 1 copy of the gene, James has 2 copies of the gene and Harry has 2 copies of the gene, the probability that Lily gives one copy of the gene is `0.5` and James give one copy of the gene is `0.99`. Therefore, the probability that Harry has 2 copies of the gene would have been `0.5 * 0.99 = 0.495`.

Implementation Hints

Recall that to compute a joint probability of multiple events, you can do so by multiplying those probabilities together. But remember that for any child, the probability of them having a certain number of genes is conditional on what genes their parents have.

Submission Guidelines

Each dataset has people that don't have any value for the trait. You should not need to modify anything else in `heredity.py` other than the three functions the specification calls for you to implement, though you may write additional functions and/or import other Python standard library modules. You may also import `numpy`, if familiar with it, but you should not need to use any other third-party Python modules.