Software Lab 09 *6.01 – Fall 2015*
Probability Distributions

Goals: To better understand probability distributions and the basic operations on them by implementing them in Python.

1 Getting Started

This lab can be done on your own, or with a partner of your choosing.

Get today's files by running:

```
> athrun 6.01 getFiles
```

or from the **online version**[1] of this lab.

Note that there are optional practice problems available from the online version of the lab, to help you if you get stuck. This lab will refer to these exercises at several points.

2 DDist

This week, we will implement classes and methods to represent and manipulate discrete probability distributions. Our starting point will be the class `DDist`, which represents a probability distribution over a discrete set. A code skeleton for `DDist` can be found in `dist.py` in this week's distribution. At initialization time, our class will take as input a dictionary, whose keys are members of the sample set of the distribution, and whose values are the associated probabilities. The `__init__` method, which we have implemented for you, checks whether the distribution is valid, and stores the input dictionary as an attribute `self.d`. Note that we could have chosen other representations, and we will try to minimize the impact of this choice on the outward-facing pieces of `DDist`. Over the course of the lab, we will implement several features in the class `DDist`, but the whole of `DDist` will only be checked once, in the last tutor problem associated with the lab. You should be prepared to construct some test cases of your own to make sure these features are working correctly, as that tutor problem will not allow enough checks to double-check each subpart.

[1] `https://sicp-s4.mit.edu/tutor/6.01/SL09`

3 Basic Functionality

In this part of the lab, we will be adding two methods to `DDist`, `prob` and `support`:

- `prob` should take one input, representing an element, and should return a single number representing the probability assigned to it by the distribution. If an element `e` is not explicitly represented in the distribution `d`, then `d.prob(e)` should return `0`.
- `support` should take no inputs, and should return the *support* of this distribution, as a list; that is, it should return a list of all elements whose probability in the distribution is nonzero.

Check Yourself 1. Implement `prob` and `support`, and construct some test cases of your own to make sure they work as intended.

4 Conditional and Joint Distributions

Note: for the rest of the lab, your code should not rely on the fact that we are using a dictionary as our internal representation; the rest of the procedures can and should be implemented without accessing the internal dictionary directly.

Some information on representation before we move on:

- We will represent the conditional probability distribution $\Pr(A|B)$ as a procedure that takes a value b of random variable B as input and returns a distribution over A, $\Pr(A|B = b)$, as an instance of `DDist`.
- We will represent the joint probability distribution $\Pr(A, B)$ as an instance of `DDist` over pairs (a, b), where a is a value of A and b is a value of B. In Python, these pairs must be represented as tuples `(a,b)`, rather than lists `[a,b]`[2].

Check Yourself 2. Consult the lab appendix on distributions and their Pythonic representations, and ask for help if you do not understand.

4.1 Joint Distributions

In this section, we will implement a stand-alone procedure `makeJointDistribution`, which takes as arguments a `DDist` representing $\Pr(A)$, and a function representing $\Pr(B \mid A)$, and returns an instance of `DDist` representing the joint distribution $\Pr(A, B)$, with A and B specifically in that order.

Tutor Question: Implement and test `makeJointDistribution`, and then paste your code into the tutor. If you are having trouble, see the practice problem about **joint distributions**.

[2] This is because these pairs will be used as keys in the dictionary insde `DDist`, and dictionary keys must not be mutable. Tuples are not mutable (i.e., you cannot change an element inside a tuple) but lists are.

5 Projection and Conditioning

In this section, we will implement two more methods inside `DDist`: `project` and `condition`

5.1 Projection

Assuming that `self` represents a distribution $\Pr(A)$, `project` should take as its lone argument a function f and return a new `DDist` representing a distribution $\Pr_{new}(A')$ over all possible $a' = f(a)$, such that:

$$\Pr_{new}(A' = a') = \sum_{a\,:\,f(a)=a'} \Pr(A = a)$$

The idea is that we are taking the elements a in the support of A and projecting them into a new domain, where the elements are $f(a)$ for all $a \in A$.

5.1.1 Projection Example

For example, imagine we have a distribution over a random variable X with integer support:

```
>>> pX = DDist({-2: .1, 2: .3, -1: .4, 1: .1, 3: .1})
```

We could project this distribution into a new domain to get the distribution over X^2:

```
>>> pXSquared = pX.project(lambda x: x**2)
>>> print pXSquared
DDist({4:0.4, 1:0.5, 9:0.1})
```

Check Yourself 3. Implement `project` in your `DDist` class. Run some tests to make sure it behaves as intended. If you are having trouble, see the practice problem on **projection**.

5.1.2 Marginalize

Define a procedure `marginalize(d, i)`, where `d` is an instance of `DDist` whose domain is a set of fixed-length tuples and `i` is an index between `0` and `n` (not including `n`), where `n` is the length of the tuples. It should return a new instance of `DDist` with the `i`th elements marginalized out. It should behave as follows:

```
>>> marginalize(DDist({(1, 1, 1) : 0.2, (1, 1, 0) : 0.4, (1, 0, 1) : 0.4}), 2)
DDist({(1, 1) : 0.6, (1, 0) : 0.4})
```

You must use `project` in your definition.

Tutor Question: Paste your definition for `marginalize` into the tutor.

5.2 Conditioning

Assuming that `self` represents a distribution $\mathtt{Pr}(A)$, `condition` should take as its lone argument a function f (which maps from elements to boolean values), and return `DDist` representing a new, re-normalized distribution $\mathtt{Pr}_{new}(A)$ over only those elements a such that $f(a)$ returns `True`:

$$\mathtt{Pr}_{new}(A = a) = \begin{cases} \mathtt{Pr}(A = a)/Z, & \text{if } f(a) = \text{True} \\ 0, & \text{otherwise} \end{cases}$$

where Z is the sum of $\mathtt{Pr}(A = a)$ for every a in the support of $\mathtt{Pr}(A)$ such that $f(a_i) = \text{True}$.

Check Yourself 4. Implement `condition` in your `DDist` class and test it to make sure it works as expected.

Tutor Question: Paste your complete definition for the `DDist` class into the tutor.

6 Total Probability and Bayes' Rule

Now, we will implement the Law of the Total Probability and Bayes' Rule. Next week, we will make use of these methods to implement probabilistic state estimation, but for now, they are interesting and useful on their own.

6.1 Total Probability

We will implement the Law of Total Probability as a stand-alone method `totalProbability`. `totalProbability` should take two arguments: a `DDist` representing $\mathtt{Pr}(A)$, and a function representing $\mathtt{Pr}(B \mid A)$. It should return an instance of `DDist` representing $\mathtt{Pr}(B)$.

Check Yourself 5. Implement the `totalProbability` function. Your code should should make use of `makeJointDistribution`, `project`, `marginalize`, and/or `condition` as necessary. If you are having trouble, see the practice problem on **total probability**.

Tutor Question: Paste your definition for `totalProbability` into the tutor.

6.2 Bayes' Rule

We will implement Bayes' Rule as a stand-alone function `bayesRule`. It should take three arguments: an instance of `DDist` representing $P(A)$, a function representing the conditional distribution $P(B|A)$, and a specific value of b. It should return an instance of `DDist` representing $P(A|B = b)$.

Check Yourself 6. Implement the `bayesRule` function. Your code should make use of `makeJointDistribution`, `project`, `marginalize`, and/or `condition` as necessary.

Tutor Question: Paste your definition for `bayesRule` into the tutor.

7 Appendix: Notes on Distributions

7.1 Distribution

- Function from elements a of domain A into probabilities
- Math: whole distribution : $\Pr(A)$
- Math: probability of element : $\Pr(A = a)$
- Python: whole distribution : `PA = DDist({'a1' : 0.1, 'a2' : 0.9})`
- Python: probability of element : `PA.prob('a2')`

7.2 Conditional Distribution

- Function from elements b of domain B into distributions over A
- Math: whole conditional distribution : $\Pr(A \mid B)$
- Math: distribution conditioned on $B = b$: $\Pr(A \mid B = b)$
- Math: probability of element conditioned on $B = b$: $\Pr(A = a \mid B = b)$
- Python: whole distribution

```
def PAgB(b):
    if b == 'foo':
        return DDist({'a1' : 0.2, 'a2' : 0.8})
    elif b == 'bar':
        return DDist({'a3' : 0.4, 'a2' : 0.6})
    else:
        print 'Error:', b, 'not in domain of PAgB'
```

- Python: distribution conditioned on `b`: `PAgB(b)`
- Python: probability of element conditioned on `b`: `PAgB(b).prob(a)`

7.3 Joint Distribution

- Probability distribution over pairs of elements (which may themselves have more complex structure.)
- Math: $\Pr(A, B)$
- Math: $\Pr(A = a, B = b) = \Pr(B = b \mid A = a)\Pr(A = a)$
- Python: whole distribution : `PAB = DDist({('a1', 'b2') : 0.1, ('a2', 'b1') : 0.5, ('a2', 'b3') : 0.4})`
- Python: whole distribution: `makeJointDistribution(PA,PBgA)`
- Python: probability of element: `PAB.prob(('a1', 'b2'))`

7.4 Projection

- Given a probability distribution over elements in one space, find a related distribution over functions of those elements.
- Math: Old random var A; New random var $B = f(A)$
- Math: Element of distribution $\Pr(B = b) = \sum_{a:f(a)=b} \Pr(A = a)$
- Python: `PA.project(f)`

7.5 Marginalization

- Given a probability distribution over elements in a joint distribution, project into a distribution on one of the dimensions.
- Math: Go from $\Pr(A, B)$ to $\Pr(A) = \sum_b \Pr(A, B)$
- Python: `PA = PAB.project(m)` (determining `m` is a tutor problem.)

7.6 Conditioning

- Given a joint probability distribution, condition on one random variable having a particular value
- Math: Go from joint $\Pr(A, B)$ and evidence b to whole distribution $\Pr(A \mid B = b)$; this is a distribution on A
- Math: Individual elements of the distribution

$$\Pr(A = a \mid B = b) = \frac{\Pr(A = a, B = b)}{\Pr(B = b)}$$

- Python: whole distribution `PAgb = PAB.condition(lambda ab : ab[1] == b)`

7.7 Total Probability

- Given a conditional distribution and a marginal, compute the other marginal
- Math: Go from $\Pr(B \mid A)$ and $\Pr(A)$ to $\Pr(B)$
- Math: Individual elements

$$\Pr(B = b) = \sum_a \Pr(A = a, B = b) = \sum_a \Pr(B = b \mid A = a) \Pr(A = a)$$

- Python: whole distribution : `PB = totalProbability(PA, PBgA)`

7.8 Bayes' Rule

- Use conditional probability to switch the conditioning relationship
- Math: Go from $\Pr(H)$ and $\Pr(E \mid H)$ to $\Pr(H \mid E = e)$
- Math: You should be able to do the derivation using definition of conditional probability
- Python: `PHge = bayesRule(PH, PEgH, e)`