Problem Wk.10.1.6: Implementing Joint distributions

Part 1: Creating a joint distribution

Write the $\text{JDist}$ function, which will represent the joint probability distribution for two variables $A$ and $B$. The function is given as inputs $\text{PA}$, which is a $\text{DDist}$ representing the probability distribution for $A$, and $\text{PBgA}$, which is a function that takes a value of $A$ as input and returns a $\text{DDist}$ representing the conditional probability of $B$ given $A$ (i.e., $P(B \mid A)$). Consider the following example:

```python
def PTgD(val):
    if val == 'disease':
        return dist.DDist({'posTest': 0.9, 'negTest': 0.1})
    else:
        return dist.DDist({'posTest': 0.5, 'negTest': 0.5})

>>> disease = dist.DDist({'disease': 0.1, 'noDisease': 0.9})
>>> dist.JDist(disease, PTgD)
DDist((noDisease, posTest): 0.450000, (disease, posTest): 0.090000,
     (noDisease, negTest): 0.450000, (disease, negTest): 0.010000)
```

$\text{JDist}$ should return a $\text{DDist}$ whose items are tuples of the form $(a,b)$, with values that are the corresponding joint probabilities. In the tuple $(a,b)$, $a$ is an element in the support of $P(A)$ and $b$ is an element in the support of $P(B \mid A)$.

$\text{JDist}$ is in the $\text{dist}$ module, so it has access to $\text{DDist}$ directly. So, in your solution, you do not need to use $\text{dist.DDist}$; you can just $\text{DDist}$.

```python
def JDist(PA, PBgA):
    pass
```

Part 2: Implement marginalization

Add the $\text{marginalizeOut}$ method to the $\text{DDist}$ class; index is an integer specifying which of the variables to marginalize out; if index is 0, then marginalize out the first variable, if index is 1, then marginalize out the second variable. The method should return a $\text{DDist}$.
Continuing the example in the previous part of this problem:

```python
global
>>> JDist(disease, PTgD).marginalizeOut(0)
DDist(postTest: 0.540000, negTest: 0.460000)
```

The following function is already defined:

```python
def removeElt(items, i):
    result = items[:i] + items[i+1:]
    if len(result) == 1:
        return result[0]
    else:
        return result
```

This removes the $i$th entry from a tuple. However, if the resulting tuple has a single element, it just returns that element, instead of a tuple of one element.

The following function is also defined:

```python
def incrDictEntry(d, k, v):
    if d.has_key(k):
        d[k] += v
    else:
        d[k] = v
```

This increments the key $k$ in dictionary $d$ by the value $v$. If the key does not already exist, it behaves as if the key existed with the value 0.

```python
class DDist:
    def __init__(self, dictionary):
        self.d = dictionary
    def prob(self, elt):
        if self.d.has_key(elt):
            return self.d[elt]
        else:
            return 0
    def support(self):
        return [k for k in self.d.keys() if self.prob(k) > 0]
    def marginalizeOut(self, index):
        pass
```

**Part 3: Implement conditioning**

When we condition on one of the random variables having a particular value, that variable becomes irrelevant and need not be represented. Conditioning effectively selects a row or a column of the joint distribution and normalizes it appropriately. Implement the `conditionOnVar(self, index, value)` method of the `DDist` class; `index` is 0 or 1, determining which variable is being specified, and `value` gives the value that variable is being assumed to have. The method should return a `DDist` over the variable
that was left unspecified. Continuing the example:

```python
>>> JDist(disease, PTgD).conditionOnVar(1, 'posTest')
DDist(noDisease: 0.833333, disease: 0.166667)
```

The `removeElt` function is also defined for this problem.

class DDist:
    def __init__(self, dictionary):
        self.d = dictionary
    def prob(self, elt):
        if self.d.has_key(elt):
            return self.d[elt]
        else:
            return 0
    def support(self):
        return [k for k in self.d.keys() if self.prob(k) > 0]
    def conditionOnVar(self, index, value):
        pass
6.01SC Introduction to Electrical Engineering and Computer Science
Spring 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.