Laporan Probability

Nama : Alif As'ad Ramadhan

NRP : 5054231007

Siapkan Library dan Source

```
# Import wanring untuk mengabaikan Warning
import warnings
warnings.filterwarnings("ignore")

!pip install probability
from probability import *
from utils import print_table
from notebook import psource, pseudocode, heatmap
```

```
Requirement already satisfied: probability in c:\users\lenovo\
anaconda3\lib\site-packages (0.0.161)
Requirement already satisfied: matplotlib in c:\users\lenovo\
anaconda3\lib\site-packages (from probability) (3.8.4)
Requirement already satisfied: mpl-format in c:\users\lenovo\
anaconda3\lib\site-packages (from probability) (0.318)
Requirement already satisfied: networkx in c:\users\lenovo\anaconda3\
lib\site-packages (from probability) (3.2.1)
Requirement already satisfied: numba in c:\users\lenovo\anaconda3\lib\
site-packages (from probability) (0.59.1)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\lib\
site-packages (from probability) (1.26.4)
Requirement already satisfied: pandas in c:\users\lenovo\anaconda3\
lib\site-packages (from probability) (2.2.2)
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\lib\
site-packages (from probability) (1.13.1)
Requirement already satisfied: seaborn in c:\users\lenovo\anaconda3\
lib\site-packages (from probability) (0.13.2)
Requirement already satisfied: setuptools in c:\users\lenovo\
anaconda3\lib\site-packages (from probability) (69.5.1)
Requirement already satisfied: tqdm in c:\users\lenovo\anaconda3\lib\
site-packages (from probability) (4.66.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib->probability) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib->probability) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib->probability) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib->probability) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\lenovo\
```

```
anaconda3\lib\site-packages (from matplotlib->probability) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\lenovo\anaconda3\
lib\site-packages (from matplotlib->probability) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from matplotlib->probability) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\
lenovo\anaconda3\lib\site-packages (from matplotlib->probability)
(2.9.0.post0)
Requirement already satisfied: celluloid in c:\users\lenovo\anaconda3\
lib\site-packages (from mpl-format->probability) (0.2.0)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in c:\users\
lenovo\anaconda3\lib\site-packages (from numba->probability) (0.42.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\lenovo\
anaconda3\lib\site-packages (from pandas->probability) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\lenovo\
anaconda3\lib\site-packages (from pandas->probability) (2023.3)
Requirement already satisfied: colorama in c:\users\lenovo\anaconda3\
lib\site-packages (from tqdm->probability) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\anaconda3\
lib\site-packages (from python-dateutil>=2.7->matplotlib->probability)
(1.16.0)
```

Probability Distribution

Distribusi probabilitas adalah konsep penting dalam statistik dan teori probabilitas yang menggambarkan kemungkinan dari setiap nilai yang mungkin dari sebuah variabel acak. Dalam konteks distribusi probabilitas diskret, kita bekerja dengan variabel acak yang memiliki nilai-nilai diskrit atau terpisah, dan setiap nilai tersebut memiliki probabilitas tertentu.

Kelas `ProbDist` digunakan untuk mendefinisikan distribusi probabilitas diskret. Prosesnya dimulai dengan penamaan variabel acak, kemudian menetapkan probabilitas untuk setiap nilai yang mungkin dari variabel tersebut. Cara ini mirip dengan penggunaan kamus (dictionary) di mana kunci (key) merupakan nilai dari variabel acak, dan kita menetapkan probabilitas pada nilai tersebut.

Dalam implementasi ini, metode-metode khusus atau "magic methods" seperti `__getitem__` dan `__setitem__` digunakan untuk menyimpan probabilitas di dalam atribut `prob` dari objek. Hal ini memungkinkan kita untuk mengakses dan menetapkan probabilitas dengan mudah, sama seperti kita mengakses atau menetapkan nilai pada kamus.

Dengan membuka kode sumber sambil menjalankan kode lainnya, kita dapat memperoleh pemahaman yang lebih mendalam tentang bagaimana distribusi probabilitas diskret diimplementasikan dalam kelas `ProbDist`.

```
psource(ProbDist)
```

```
<IPython.core.display.HTML object>
```

```
class ProbDist:
    """A discrete probability distribution. You name the random variable
    in the constructor, then assign and query probability of values.
    >>> P = ProbDist('Flip'); P['H'], P['T'] = 0.25, 0.75; P['H']
    0.25
    >>> P = ProbDist('X', {'lo': 125, 'med': 375, 'hi': 500})
    >>> P['lo'], P['med'], P['hi']
    (0.125, 0.375, 0.5)
    """

    def __init__(self, varname='?', freqs=None):
        """If freqs is given, it is a dictionary of values - frequency pairs,
        then ProbDist is normalized."""
        self.prob = {}
        self.varname = varname
        self.values = []
        if freqs:
            for (v, p) in freqs.items():
                self[v] = p
            self.normalize()

    def __getitem__(self, val):
        """Given a value, return P(value)."""
        try:
            return self.prob[val]
        except KeyError:
            return 0

    def __setitem__(self, val, p):
        """Set P(val) = p."""
        if val not in self.values:
            self.values.append(val)
        self.prob[val] = p

    def normalize(self):
        """Make sure the probabilities of all values sum to 1.
        Returns the normalized distribution.
        Raises a ZeroDivisionError if the sum of the values is 0."""
        total = sum(self.prob.values())
        if not isclose(total, 1.0):
            for val in self.prob:
                self.prob[val] /= total
        return self

    def show_approx(self, numfmt='{:.3g}'):
        """Show the probabilities rounded and sorted by key, for the
        sake of portable doctests."""
        return ', '.join([('{}: ' + numfmt).format(v, p)
                          for (v, p) in sorted(self.prob.items())])

    def __repr__(self):
        return "P({})".format(self.varname)
```

Fungsi-fungsi utama dalam kelas ProbDist:

- `__init__`: Menginisialisasi objek ProbDist dengan nama variabel acak (varname) dan nilai frekuensi (freqs). Jika freqs diberikan, kelas akan otomatis menormalisasi probabilitas.
- `__getitem__`: Mengembalikan probabilitas untuk nilai yang diberikan.
- `__setitem__`: Menetapkan probabilitas untuk nilai tertentu. Jika nilai tersebut baru, maka akan ditambahkan ke daftar values.
- `normalize`: Menormalkan probabilitas sehingga totalnya menjadi 1.

- `show_approx`: Menampilkan probabilitas yang telah dibulatkan ke tiga desimal untuk memudahkan pembacaan.

Contoh Penggunaan

Mendefinisikan Distribusi Probabilitas

```
p = ProbDist("Flip")
p["H"], p["T"] = 0.25, 0.75
p["H"]
```

```
0.25
```

Di sini, dibuat distribusi untuk variabel acak "Flip" dengan dua nilai, "H" dan "T" (kepala dan ekor), dengan probabilitas masing-masing 0.25 dan 0.75.

Menggunakan Parameter `freq`

Parameter `freqs` merupakan sebuah dictionary yang berisi pasangan nilai dan frekuensi dari variabel acak yang akan digunakan untuk membangun distribusi probabilitas. Nilai freqs ini secara otomatis dinormalisasi oleh kelas `ProbDist` sehingga total probabilitasnya mencapai 1.

```
p = ProbDist(freq={'low': 125, 'medium': 375, 'high': 500})
(p['low'], p['medium'], p['high'])
```

```
(0.125, 0.375, 0.5)
```

Parameter `varname` adalah parameter pertama dalam konstruktor `ProbDist`, yang digunakan untuk memberikan nama pada variabel acak. Jika tidak ada nama yang diberikan, nilai default dari varname adalah simbol `?`. Dalam contoh pada gambar, kita membuat objek ProbDist tanpa menentukan nama variabel, sehingga `p.varname` menghasilkan `?`.

```
p.var_name
```

```
'?'
```

Menambahkan Nilai Secara Bertahap

Kita juga bisa menambahkan nilai baru secara bertahap. Setiap kali nilai baru ditambahkan melalui metode `__setitem__`, nilai tersebut ditambahkan ke dalam daftar values.

```
p.values
```

```
['low', 'medium', 'high']
```

Normalisasi dan Penggunaan `show_approx`

Jika nilai baru ditambahkan secara bertahap, distribusi tidak akan otomatis dinormalisasi. Kita dapat memanggil metode normalize untuk menormalkan nilai probabilitas. Fungsi

`show_approx` berguna untuk menampilkan probabilitas dalam format yang lebih mudah dibaca, dengan pembulatan desimal.

```
p = ProbDist("Y")
p["Cat"] = 50
p["Dog"] = 114
p["Mice"] = 64
p.normalize()
p.show_approx()
```

```
'Cat: 0.219, Dog: 0.5, Mice: 0.281'
```

Joint Probability Distribution dan Fungsi

Dalam distribusi probabilitas bersama (Joint Probability Distribution), kita bekerja dengan probabilitas dari beberapa variabel acak secara bersamaan. Untuk mendapatkan nilai dari setiap variabel acak dalam suatu kejadian, kita dapat menggunakan fungsi bantu `event_values`.

Cara Kerja event_values

- Input: Fungsi ini menerima dua argumen:
  - event: Dictionary yang berisi pasangan nama variabel dan nilainya.
  - variables: List yang berisi nama-nama variabel yang nilainya akan diambil dari event.
- Output: Fungsi akan mengembalikan tuple yang berisi nilai-nilai dari variabel-variabel dalam variables, sesuai dengan urutan dalam variables.

```
event = {'A': 10, 'B': 9, 'C': 8}
variables = ['C', 'A']
event_values(event, variables)
```

```
(8, 10)
```

Model probabilitas sepenuhnya ditentukan oleh distribusi gabungan untuk semua variabel acak. Modul probabilitas mengimplementasikannya sebagai kelas JointProbDist yang mewarisi kelas ProbDist. Kelas ini menentukan distribusi probabilitas diskrit pada sekumpulan variabel.

```
psource(JointProbDist)
```

```
<IPython.core.display.HTML object>
```

```
class JointProbDist(ProbDist):
    """A discrete probability distribute over a set of variables.
    >>> P = JointProbDist(['X', 'Y']); P[1, 1] = 0.25
    >>> P[1, 1]
    0.25
    >>> P[dict(X=0, Y=1)] = 0.5
    >>> P[dict(X=0, Y=1)]
    0.5"""

    def __init__(self, variables):
        self.prob = {}
        self.variables = variables
        self.vals = defaultdict(list)

    def __getitem__(self, values):
        """Given a tuple or dict of values, return P(values)."""
        values = event_values(values, self.variables)
        return ProbDist.__getitem__(self, values)

    def __setitem__(self, values, p):
        """Set P(values) = p. Values can be a tuple or a dict; it must
        have a value for each of the variables in the joint. Also keep track
        of the values we have seen so far for each variable."""
        values = event_values(values, self.variables)
        self.prob[values] = p
        for var, val in zip(self.variables, values):
            if val not in self.vals[var]:
                self.vals[var].append(val)

    def values(self, var):
        """Return the set of possible values for a variable."""
        return self.vals[var]

    def __repr__(self):
        return "P({})".format(self.variables)
```

Joint Probability Distribution dengan Kelas `JointProbDist`

Distribusi probabilitas bersama (Joint Probability Distribution) adalah distribusi yang menghubungkan probabilitas dari beberapa variabel acak yang berhubungan, di mana setiap kombinasi nilai dari variabel-variabel tersebut memiliki probabilitas tertentu.

`variables` adalah daftar yang berisi nama-nama variabel `X` dan `Y`. Kita kemudian membuat objek JointProbDist bernama `j`, yang akan menyimpan distribusi probabilitas bersama untuk X dan Y.

```
variables = ['X', 'Y']
j = JointProbDist(variables)
j
```

```
P(['X', 'Y'])
```

Kelas JointProbDist mirip dengan kelas ProbDist karena keduanya menggunakan metode-metode "magic" untuk menetapkan probabilitas pada nilai-nilai tertentu. Probabilitas untuk kombinasi nilai variabel dapat ditetapkan dalam dua format:

- Dengan memberikan tuple: `j[1, 1] = 0.2` berarti probabilitas `P(X=1, Y=1)` adalah `0.2`.
- Dengan menggunakan dictionary: `j[dict(X=0, Y=1)] = 0.5` berarti probabilitas `P(X=0, Y=1)` adalah `0.5`.
- Fungsi `event_values` digunakan di dalam metode `__getitem__` dan `__setitem__` untuk memproses format nilai ini sehingga bisa bekerja dengan kedua jenis input tersebut.

```
j[1,1] = 0.2
j[dict(X=0, Y=1)] = 0.5
```

```
(j[1,1], j[0,1])
```

```
(0.2, 0.5)
```

Kita juga dapat menggunakan metode values untuk mendapatkan semua nilai yang mungkin untuk variabel tertentu dalam distribusi bersama. Misalnya:

```
j.values('X')
```

```
[1, 0]
```

Inference Menggunakan Full Joint Distributions

Pada bagian ini, kita akan menggunakan Full Joint Distributions untuk menghitung distribusi posterior berdasarkan beberapa evidence (bukti) yang diberikan. Distribusi posterior dihitung dengan mempertimbangkan nilai dari variabel yang diketahui dan variabel yang ingin kita estimasi.

1. Representasi Evidence

Evidence atau bukti direpresentasikan menggunakan dictionary di Python, di mana:

```
- Key dalam dictionary adalah nama variabel.
```

```
- Value dalam dictionary adalah nilai variabel tersebut.
```

```
evidence = {'Cavity': True, 'Toothache': False}
```

Sebagai contoh, kita akan menggunakan distribusi probabilitas bersama penuh (full joint distribution) yang menggambarkan tiga variabel acak: `Cavity`, `Toothache`, dan `Catch`. Berikut adalah distribusi probabilitas yang diberikan:

```
full_joint = JointProbDist(['Cavity', 'Toothache', 'Catch'])
full_joint[dict(Cavity=True, Toothache=True, Catch=True)] = 0.108
full_joint[dict(Cavity=True, Toothache=True, Catch=False)] = 0.012
full_joint[dict(Cavity=True, Toothache=False, Catch=True)] = 0.016
full_joint[dict(Cavity=True, Toothache=False, Catch=False)] = 0.064
full_joint[dict(Cavity=False, Toothache=True, Catch=True)] = 0.072
full_joint[dict(Cavity=False, Toothache=False, Catch=True)] = 0.144
full_joint[dict(Cavity=False, Toothache=True, Catch=False)] = 0.008
full_joint[dict(Cavity=False, Toothache=False, Catch=False)] = 0.576
```

Fungsi `enumerate_joint` menghitung jumlah dari semua entri dalam distribusi `P` yang konsisten dengan evidence `e`. Parameter variables pada fungsi ini merujuk ke variabel-variabel sisa (yang tidak ada dalam e). Fungsi ini menggunakan rekursi, di mana pada setiap panggilan rekursif, fungsi mempertahankan satu variabel tetap konstan sambil memvariasikan yang lain.

Secara umum, langkah kerja dari enumerate_joint adalah sebagai berikut:

- Fungsi menerima variables (variabel yang tersisa untuk dihitung).
- Fungsi akan memproses secara rekursif, menjaga satu variabel konstan pada setiap iterasi sambil memvariasikan nilai variabel lainnya hingga semua kombinasi terhitung.

```
psource(enumerate_joint)
```

```
<IPython.core.display.HTML object>
```

```
def enumerate_joint(variables, e, P):
    """Return the sum of those entries in P consistent with e,
    provided variables is P's remaining variables (the ones not in e)."""
    if not variables:
        return P[e]
    Y, rest = variables[0], variables[1:]
    return sum([enumerate_joint(rest, extend(e, Y, y), P) for y in P.values(Y)])
```

Menghitung Probabilitas Marginal `P(Toothache=True)`

Untuk menghitung probabilitas marginal P (Toothache=True) ,kita bisa menggunakan marginalisasi. Proses marginalisasi ini menghitung total probabilitas semua nilai yang konsisten dengan evidence yang diberikan. Di sini kita set `Toothache=True` sebagai `evidence`, sedangkan variabel lainnya, yaitu `Cavity` dan `Catch`, merupakan variabel yang tidak dihitung.

```
evidence = dict(Toothache=True)
variables = ['Cavity', 'Catch'] # variable lainnya
ans1 = enumerate_joint(variables, evidence, full_joint)
ans1
```

```
0.19999999999999998
```

Menghitung Probabilitas Bersama `P(Cavity=True and Toothache=True)`

Kita juga dapat menggunakan fungsi yang sama untuk menghitung probabilitas bersama, misalnya P `( Cavity=True and Toothache=True )`. Dalam kasus ini, kita set evidence dengan nilai `Cavity=True` dan `Toothache=True`, sedangkan variabel `Catch` tidak dihitung.

```
evidence = dict(Cavity=True, Toothache=True)
variables = ['Catch'] # variabel lainnya
ans2 = enumerate_joint(variables, evidence, full_joint)
ans2
```

```
0.12
```

Dengan menghitung probabilitas marginal dan probabilitas bersama, kita bisa menghitung probabilitas kondisional P `( Cavity=True | Toothache=True )` menggunakan rumus:

$$P(Cavity = True|Toothache = True) = \frac{P(Cavity = True\ and\ Toothache = True)}{P(Toothache = True)}$$

Karena kita telah menghitung nilai untuk pembilang (numerator) dan penyebut (denominator), kita bisa melakukan pembagian:

```
ans2/ans1
```

```
0.6
```

Ini menunjukkan bahwa probabilitas kondisional `P(Cavity=True | Toothache=True) = 0.6`. Dengan demikian, jika seseorang mengalami sakit gigi `(Toothache=True)`, ada kemungkinan `60%` bahwa mereka memiliki masalah gigi berlubang `(Cavity=True)`.

Dalam analisis probabilitas, kita sering kali tertarik untuk mengetahui distribusi probabilitas dari variabel tertentu yang dipengaruhi oleh bukti tertentu. Hal ini sering kali melibatkan perhitungan untuk setiap kemungkinan nilai dari variabel tersebut. Salah satu metode yang digunakan untuk melakukan ini adalah dengan menggunakan fungsi `enumerate_joint_ask`, yang memberikan distribusi probabilitas terhadap nilai-nilai variabel X, dengan memperhitungkan pengamatan yang telah diberikan.

```
psource(enumerate_joint_ask)
```

```
<IPython.core.display.HTML object>
```

```
def enumerate_joint_ask(X, e, P):
    """
    [Section 13.3]
    Return a probability distribution over the values of the variable X,
    given the {var:val} observations e, in the JointProbDist P.
    >>> P = JointProbDist(['X', 'Y'])
    >>> P[0,0] = 0.25; P[0,1] = 0.5; P[1,1] = P[2,1] = 0.125
    >>> enumerate_joint_ask('X', dict(Y=1), P).show_approx()
    '0: 0.667, 1: 0.167, 2: 0.167'
    """
    assert X not in e, "Query variable must be distinct from evidence"
    Q = ProbDist(X)  # probability distribution for X, initially empty
    Y = [v for v in P.variables if v != X and v not in e]  # hidden variables.
    for xi in P.values(X):
        Q[xi] = enumerate_joint(Y, extend(e, X, xi), P)
    return Q.normalize()
```

kita dapat menggunakan fungsi ini untuk menghitung probabilitas P `( Cavity | Toothache=True )`. Berikut adalah langkah-langkah dalam penerapan:

```
query_variable = 'Cavity'
evidence = dict(Toothache=True)
ans = enumerate_joint_ask(query_variable, evidence, full_joint)
(ans[True], ans[False])
```

```
(0.6, 0.39999999999999997)
```

Hasil dari perhitungan ini adalah P `( Cavity=True )` sebesar 0.6 dan P `( Cavity=False )` sebesar 0.4. Hasil ini konsisten dengan perhitungan manual yang dilakukan sebelumnya.