Laporan Uji Coba Praktikum Probabilistic Learning Kecerdasan Komputasional (KK) - RKA Kelas N

Nama : Abdan Hafidz

NRP : 5054231021

In [1]:
```
from probability import *
from utils import print_table
from notebook import psource, pseudocode, heatmap
```

Distribusi Peluang

Kelas `ProbDist` mendefinisikan distribusi probabilitas diskrit. Pada kelas ini, kita memberi nama untuk variabel acak dan menetapkan probabilitas pada setiap nilai yang mungkin diambil oleh variabel acak tersebut. Ini memungkinkan kita menghitung probabilitas suatu nilai spesifik dalam rangkaian nilai yang mungkin.

In [ ]:
```
psource(ProbDist)
```

Misalkan kita ingin mendefinisikan distribusi Peluang besok ada kelas atau tidak

In [4]:
```
p = ProbDist('Kelas')
p['Ada'], p['TidakAda'] = 0.50, 0.50
p['TidakAda']
```

Out[4]:
```
0.5
```

Sebagai kasus lain kita mempunyai contoh yaitu kita ingin menghitung peluang mengambil sebuah bola di antara bola merah sebanyak 100 bola, kuning sebanyak 50 bola, dan hijau sebanyak 20 bola **(asumsi bola yang sudah diambil dikembalikan lagi)**.

In [11]:
```
p_ambil_bola = ProbDist(freq={'red': 100, 'yellow': 50, 'green': 20})
```

Secara umum kita dapat menentukan peluang termabil 1 bola merah adalah 100/170 = 10/17 , 1 bola kuning = 50/170 = 5/17, dan 1 bola hijau adalah 20/170 = 2/17, dengan menggunakan `ProbDist` program akan menghitung peluang - peluang tersebut secara otomatis berdasarkan distribusi frekuensi yang disediakan.

In [15]:
```
print("Peluang Terambil 1 bola merah : ")
print(p_ambil_bola['red'])
print("Peluang Terambil 1 bola kuning : ")
print(p_ambil_bola['yellow'])
```

```
print("Peluang Terambil 1 bola hijau : ")
print(p_ambil_bola['green'])
```

```
Peluang Terambil 1 bola merah :
0.5882352941176471
Peluang Terambil 1 bola kuning :
0.29411764705882354
Peluang Terambil 1 bola hijau :
0.11764705882352941
```

Kita juga dapat memeriksa kejadian apa saja yang tersedia pada sampel dengan menggunakan method `values`

In [17]:
```
p_ambil_bola.values
```

Out[17]:
```
['red', 'yellow', 'green']
```

Melakukan normalisasi nilai peluang agar terdistribusi secara normal.

In [21]:
```
p_ambil_bola.normalize
```

Out[21]:
```
<bound method ProbDist.normalize of P(?)>
```

Membulatkan nilai desimal dari nilai peluang

In [22]:
```
p_ambil_bola.show_approx()
```

Out[22]:
```
'green: 0.118, red: 0.588, yellow: 0.294'
```

Joint Probability Distribution

Function method `event_values` menghasilkan tuple dari nilai-nilai variabel dalam sebuah event. Event ini ditentukan dalam bentuk dictionary di mana kunci adalah nama variabel dan nilainya adalah nilai dari variabel tersebut. Variabel-variabel ditentukan dengan daftar, dan urutan tuple yang dihasilkan sama dengan urutan variabel tersebut.

Alternatifnya, jika event diberikan sebagai list atau tuple dengan panjang yang sama dengan variabel, maka tuple dari event akan dikembalikan apa adanya.

In [23]:
```
event = {'AmbilMerah': p_ambil_bola['red'],
         'AmbilKuning': p_ambil_bola['yellow'],
         'AmbilHijau': p_ambil_bola['green']}
variables = ['AmbilMerah', 'AmbilKuning']
event_values(event, variables)
```

Out[23]:
```
(0.5882352941176471, 0.29411764705882354)
```

Sebuah model probabilitas sepenuhnya ditentukan oleh distribusi gabungan dari semua variabel acak.Modul probabilitas mengimplementasikan hal ini melalui kelas JointProbDist

yang merupakan turunan dari kelas ProbDist. Kelas ini menentukan distribusi probabilitas diskrit pada sekumpulan variabel tertentu.

In [ ]:
```
psource(JointProbDist)
```

Nilai untuk Distribusi Gabungan adalah tuple berurutan, di mana setiap elemen mewakili nilai yang sesuai dengan variabel tertentu. Untuk Distribusi Gabungan dari variabel AmbilMerah dan AmbilKuning.

In [25]:
```
j = JointProbDist(variables)
j
```

Out[25]:
```
P(['AmbilMerah', 'AmbilKuning'])
```

Sebagai contoh kita dapat menentukan peluang terambil merah dan kuning

In [27]:
```
j[dict(AmbilMerah = 1, AmbilKuning = 1)] = p_ambil_bola['red'] *
p_ambil_bola['yellow']
j[1,1]
```

Out[27]:
```
0.17301038062283738
```

Kita juga bisa menampilkan semua nilai untuk suatu variabel tertentu menggunakan metode values.

In [28]:
```
j.values('AmbilMerah')
```

Out[28]:
```
[1]
```

Inferensi Menggunakan Distribusi Gabungan Penuh

Pada bagian ini, kita menggunakan Distribusi Gabungan Penuh untuk menghitung distribusi posterior berdasarkan beberapa bukti. Bukti ini direpresentasikan sebagai *dictionary* di Python, dengan variabel sebagai kunci dan nilai bukti sebagai nilai kunci tersebut.

$$\mathbf{P}(X|\mathbf{e}) = \alpha\mathbf{P}(X,\mathbf{e}) = \alpha\sum_{y}\mathbf{P}(X,\mathbf{e},\mathbf{y})$$

Di sini, **α** adalah faktor normalisasi, **X** adalah variabel yang ditanyakan, dan **e** adalah bukti. Berdasarkan persamaan ini, kita menjumlahkan pada variabel sisa **y** (yang bukan bagian dari bukti atau variabel yang ditanyakan), yaitu semua kombinasi kemungkinan **y**.

Contoh Kasus: Distribusi Gabungan untuk Penyakit

Mari kita buat contoh distribusi gabungan yang berhubungan dengan diagnosis penyakit berdasarkan dua gejala: **Demam** dan **Batuk**, serta hasil **Tes COVID-19**. Kita akan mendefinisikan distribusi gabungan untuk variabel-variabel ini.

In [29]:
```
full_joint = JointProbDist(['Demam', 'Batuk', 'TesCOVID'])
full_joint[dict(Demam=True, Batuk=True, TesCOVID=True)] = 0.1
full_joint[dict(Demam=True, Batuk=True, TesCOVID=False)] = 0.05
full_joint[dict(Demam=True, Batuk=False, TesCOVID=True)] = 0.07
full_joint[dict(Demam=True, Batuk=False, TesCOVID=False)] = 0.03
full_joint[dict(Demam=False, Batuk=True, TesCOVID=True)] = 0.08
full_joint[dict(Demam=False, Batuk=False, TesCOVID=True)] = 0.15
full_joint[dict(Demam=False, Batuk=True, TesCOVID=False)] = 0.02
full_joint[dict(Demam=False, Batuk=False, TesCOVID=False)] = 0.45
```

Mari kita lihat fungsi *enumerate_joint* yang mengembalikan jumlah dari entri-entri dalam distribusi P yang konsisten dengan bukti e, di mana variabel yang disediakan adalah variabel sisa dari P (yang tidak ada dalam e). Di sini, P merujuk pada distribusi gabungan penuh.

Fungsi ini menggunakan pemanggilan rekursif dalam implementasinya. Parameter pertama, *variables*, merujuk pada variabel-variabel yang tersisa. Dalam setiap pemanggilan rekursif, fungsi ini menjaga satu variabel tetap konstan sambil memvariasikan variabel lainnya.

Contoh Kasus

Pada contoh distribusi gabungan yang telah kita buat sebelumnya, kita memiliki tiga variabel: **Demam**, **Batuk**, dan **TesCOVID**. Misalkan kita ingin menghitung probabilitas seorang pasien yang mengalami demam dan batuk berdasarkan hasil tes COVID-19.

1. **Bukti (e)**: Misalkan kita ingin memasukkan bukti bahwa pasien mengalami demam dan batuk (e: `{'Demam': True, 'Batuk': True}`).
2. **Variabel yang Tersisa**: Variabel yang tersisa dalam distribusi adalah **TesCOVID**.

Fungsi *enumerate_joint* akan menjumlahkan semua entri dalam distribusi P yang konsisten dengan bukti bahwa pasien mengalami demam dan batuk, sambil memvariasikan hasil tes COVID-19. Hasil akhir dari fungsi ini memberikan kita probabilitas total berdasarkan kondisi yang diberikan, yang membantu dalam pengambilan keputusan medis.

In [ ]:
```
psource(enumerate_joint)
```

Mencari Probabilitas Marginal P(Batuk=True)

Misalkan kita ingin menemukan **P(Batuk=True)**. Kita dapat memperoleh nilai ini melalui marginalisasi. Kita akan menggunakan fungsi **enumerate_joint** untuk menyelesaikannya dengan menetapkan `Batuk=True` sebagai bukti. Fungsi **enumerate_joint** akan mengembalikan jumlah probabilitas yang konsisten dengan bukti tersebut, yaitu Probabilitas Marginal.

In [31]:
```
# Menentukan bukti
evidence = dict(Batuk=True)
```

```
# Variabel yang bukan bagian dari bukti
variables = ['Demam', 'TesCOVID']

# Menghitung probabilitas marginal menggunakan enumerate_joint
ans1 = enumerate_joint(variables, evidence, full_joint)
ans1
```

Out[31]:
```
0.25
```

Verifikasi Hasil Menggunakan Distribusi Gabungan Penuh

Kita dapat memverifikasi hasil dari definisi distribusi gabungan penuh. Kita juga dapat menggunakan fungsi yang sama untuk menemukan probabilitas yang lebih kompleks, seperti **P(Demam=True dan TesCOVID=True)**.

In [32]:
```
# Menentukan bukti untuk kasus kompleks
evidence_complex = dict(Demam=True, TesCOVID=True)

# Variabel yang bukan bagian dari bukti
variables_complex = ['Batuk']

# Menghitung probabilitas menggunakan enumerate_joint untuk probabilitas kompleks
ans_complex = enumerate_joint(variables_complex, evidence_complex, full_joint)
ans_complex
```

Out[32]:
```
0.17
```

Menghitung Probabilitas Bersyarat P(Demam=True | TesCOVID=True)

Kemampuan untuk menemukan jumlah probabilitas yang memenuhi bukti tertentu memungkinkan kita untuk menghitung probabilitas bersyarat seperti **P(Demam=True | TesCOVID=True)**. Kita dapat menulisnya sebagai:

$$P(Demam = True|TesCOVID = True) = \frac{P(Demam = True \text{ dan } TesCOVID = Tr}{P(TesCOVID = True)}$$

Kita telah menghitung baik pembilang maupun penyebutnya sebelumnya.

Penjelasan

1. **Pembilang**: Di sini, **P(Demam=True dan TesCOVID=True)** adalah probabilitas bahwa pasien mengalami demam dan hasil tes COVID-19 positif. Kita sudah menghitung nilai

ini sebelumnya menggunakan fungsi **enumerate_joint**.

2. **Penyebut**: **P(TesCOVID=True)** adalah probabilitas bahwa pasien memiliki hasil tes COVID-19 positif. Kita juga telah menghitung nilai ini sebelumnya dengan menetapkan `TesCOVID=True` sebagai bukti.

3. **Menghitung Probabilitas Bersyarat**: Dengan kedua nilai ini, kita dapat menghitung probabilitas bersyarat **P(Demam=True | TesCOVID=True)** dengan membagi pembilang dengan penyebut. Hasilnya memberikan kita peluang bahwa pasien mengalami demam, diberikan bahwa hasil tes COVID-19 mereka positif.

In [34]:
```
ans_complex/ans1
```

Out[34]:
```
0.68
```

Distribusi Probabilitas Bersyarat Menggunakan **enumerate_joint_ask**

Kita mungkin tertarik pada distribusi probabilitas dari suatu variabel tertentu yang dikondisikan pada bukti tertentu. Ini dapat melibatkan perhitungan seperti di atas untuk setiap kemungkinan nilai dari variabel tersebut. Hal ini telah diimplementasikan sedikit berbeda dengan menggunakan normalisasi dalam fungsi **enumerate_joint_ask**, yang mengembalikan distribusi probabilitas atas nilai-nilai variabel **X**, mengingat observasi {var:val} **e**, dalam **JointProbDist P**.

Penjelasan Implementasi

1. **Fungsi enumerate_joint_ask**: Fungsi ini memanggil **enumerate_joint** untuk setiap nilai dari variabel kueri. Misalnya, jika kita ingin menghitung distribusi probabilitas bersyarat untuk variabel **Demam** mengingat hasil tes COVID-19, kita akan melakukannya untuk setiap nilai dari **Demam** (True dan False).

2. **Bukti yang Diperluas**: Ketika memanggil **enumerate_joint**, fungsi ini akan menyertakan bukti yang diperluas dengan menambahkan **X = x_i**, di mana **x_i** adalah nilai yang sedang dianalisis untuk variabel kueri. Misalnya, jika kita sedang menganalisis **Demam=True**, maka bukti yang diperluas adalah `{'TesCOVID': True, 'Demam': True}`.

3. **Normalisasi**: Setelah mendapatkan distribusi untuk semua nilai variabel kueri, hasilnya dinormalisasi untuk menghasilkan distribusi probabilitas yang valid, di mana semua probabilitas menjumlah hingga 1.

Contoh Kasus: Distribusi Probabilitas **P(Demam | TesCOVID=True)**

Misalkan kita ingin mengetahui distribusi probabilitas dari **Demam** mengingat hasil tes COVID-19 positif:

In [ ]:
```
psource(enumerate_joint_ask)
```

Mencari **P(Demam | TesCOVID=True)** Menggunakan **enumerate_joint_ask**

Mari kita cari **P(Demam | TesCOVID=True)** menggunakan fungsi **enumerate_joint_ask**. Dalam hal ini, kita akan mengatur variabel kueri menjadi **Demam** dan bukti menjadi **TesCOVID=True**.

In [36]:
```
# Menentukan variabel kueri dan bukti
query_variable = 'Demam'
evidence = dict(TesCOVID=True)

# Menghitung probabilitas menggunakan enumerate_joint_ask
ans = enumerate_joint_ask(query_variable, evidence, full_joint)

# Menampilkan hasil untuk kedua nilai dari variabel kueri
(ans[True], ans[False])
```

Out[36]:
```
(0.425, 0.575)
```