Istraživanje podataka u bioinformatici - skripta za vežbe

Mart 2026

1 RSCU i CAI

Da bismo dobro razumeli gradivo koje sledi, pogledajmo sledeću analogiju:
DNK = tekst
gen = cela rečenica
kodon = reč od 3 slova
aminokiselina = značenje te reči
T,C,A,G = slova (timin, citozin, adenin, guanin)

Aminokiseline su organska jedinjenja koja sadrže amino ($-NH_2$) i karboksilnu grupu ($-COOH$) i predstavljaju osnovne gradivne elemente proteina (belančevina). One su ključne za rast, razvoj i funkcionisanje organizma, učestvujući u izgradnji mišića, tkiva, enzima i hormona. Poznato je oko 500 vrsta, ali svega 20 gradi proteine u ljudskom telu.

Različite aminokiseline imaju različite kodone koji ih predstavljaju. Jedna aminokiselina može biti predstavljena sa više kodona (jednom značenju može biti pridruženo više reči - na primer "auto","kola","automobil" predstavljaju isti objekat). Razlog za ovo je evolucijski. U slučaju da dodje do mutacije, i promeni se jedna baza u kodonu (npr. baza adenin predje u guanin u GAA, GAA → GAG), želimo da aminokiselina ostane ista, protein ostane isti i funkcionalan i da organizam ima veću šansu da preživi. U kodonu treća baza je najmanje bitna te tako GCU, GCC, GCA, GCG svi daju alanin (ovde je uracil U baza umesto timina T, jer se radi o RNK, a ne o DNK).

```
FFLLSSSSYY**CC*WLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG
```

je standardni genetski kod, sekvenca koja mapira aminokiseline na kodone koji ih predstavljaju (na 64 mogućnosti redom F = TTT, F=TTC, L=TTA,... i * su stop kodoni). Aminokiseline su: F - fenilalanin, L - leucin, S - serin, Y - tirozin, C - cistein, * - stop kodon, itd. Stop kodon signalizira kraj sinteze proteina i posledica toga je kada ribozom naidje na ove triplete, sinteza prestaje i novoformirani protein se oslobadja.

```
bases = ['T','C','A','G']
codons = [a + b + c for a in bases for b in bases for c in
    bases]
amino_acids = 'FFLLSSSSYY**CC*
    WLLLLPPPPHHQQRRRRIIIMTTTTNNKKSSRRVVVVAAAADDEEGGGG'
codon_table = dict(zip(codons, amino_acids))
```

- **zip** – spaja elemente iz više lista u parove (tuple-ove). Na primer: `zip([A,B],[1,2])` → `(A,1), (B,2)`. Funkcija pravi iterator tuple-ova gde se elementi uparuju po poziciji [**?**].
- **dict** – pravi rečnik (mapu) ključ → vrednost iz parova (key, value).
- **Šta radi kod:**
  - generiše sve moguće kodone (kombinacije T, C, A, G dužine 3)
  - ima niz aminokiselina
  - pomoću `dict(zip(...))` pravi mapu:

`kodon` → `aminokiselina`

Sledeći korak je da napravimo rečnik oblika aminokiselina -¿ lista svih kodona koji je predstavljaju.

```
amino_codons_dict = {}
for codon, amino:
    if amino not in amino_codons_dist:
        amino_codons_dict[amino] = [codon]
    else:
        amino_codons_dict[amino].append(codon)
```

Ovaj kod nam daje rečnik oblika $'F' : ['TTT','TTC']$, ...

RSCU (Relative Synonymous Codon Usage) predstavlja meru koliko često u nekom delu genoma koristimo odredjeni kodon za neku aminokiselinu u odnosu na to koliko bi se očekivalo da su svi kodoni te aminokiseline podjednako zastupljeni.

$$rscu[codon] = \frac{num_{codons} * num_{occurence}}{num_{all-codons-for-that-amino-in-this-seq}}$$

gde je:

- num_{codons} broj kodona koji kodiraju aminokiselinu koju kodira naš kodon čiji rscu računamo
- $num_{occurence}$ broj pojavljivanja našeg kodona u posmatranoj sekvenci
- $num_{all-codons-for-that-amino-in-this-seq}$ broj kodona koji se javljaju u sekvenci a kodiraju istu aminokiselinu kao naš

Primer: Aminokiselinu F kodiraju dva kodona TTT i TTC. Neka se u sekvenci (genu npr.) koju proučavamo TTT javlja 6 puta, a TTC 2 puta. U tom slučaju:

$$rscu['TTT'] = \frac{2 * 6}{6 + 2} = 12/8 = 1.5$$

RSCU koji je veći od 1 govori da se kodon javlja više puta nego što je očekivano, RSCU manji od 1 govori da se javlja manje puta, a RSCU jednak 1 je optimalan. RSCU je mera koja se koristi za analizu gena, poredjenje organizama, itd.

```
def my_RSCU(sequence):
    rscu = {}
    for codon in codons:
        rscu[codon] = 0.5
    num_occurence_codon = {}
    num_occurence_all_for_amino = {}
    for pos_codon in range(0,len(sequence)-2,3):
        found_codon = sequence[pos_codon:pos_codon+3]
        if (found_codon not in num_occurence_codon):
            num_occurence_codon[found_codon] = 1
        else:
            num_occurence_codon[found_codon]+=1
        amino = codon_table[found_codon]
        if (amino not in num_occurence_all_for_amino):
            num_occurence_all_for_amino[amino]=1
        else:
            num_occurence_all_for_amino[amino]+=1
    for codon in codons:
        if codon in num_occurence:
            amino = codon_table[codon]
            rscu[codon] = len(amino_codons_dict[amino]) *
                num_occurence_codon[codon]/
                num_occurence_all_for_amino[amino]
    return rscu
```

Primetimo da je na početku rscu za svaki kodon postavljen na 0.5, a ne na 0. To je zato što želimo da izbegnemo deljenje 0 a i da isključimo nerealističan slučaj da se neki kodon ne javlja uopšte. Veća je verovatnoća da je došlo do greške ili da sekvenca koju smo izabrali slučajno nije dovoljno ekspresivna za taj kodon. Naime, ljudski genom ima skoro 3 milijarde baza. Skupo je i resursno zahtevno da obradjujemo čitav genom: zahteva se sekvenciranje, analiza, skladištenje podataka, treba spojiti posle milijarde fragmenata, rešiti greške, analizirati varijacije, veoma je kompleksno, dobijamo previše informacija (to je ogroman broj mutacija, pri čemu ne znamo koje su bitne koje nisu, mnogo varijanti od toga je nepoznato, mnogo je bez efekta, teško za interpretaciju). Iz toga razloga obično čitamo samo jedan gen. To nije dovoljno za analizu čitavog organizma, ali jeste ako samo želimo da ispitujemo ponašanje toga gena.

CAI - Codon Adaptation Index, definiše adaptiranost korišćenih kodona u odnosu na kodone gena sa najvećom ekspresijom kod istog organizma. Relativna adaptiranost kodona predstavlja RSCU vrednost kodona u odnosu na najveću

vrednost RSCU sinonimnog kodona:
Primer: Uzmimo AAA i AAG. Neka je AAA kodon sa najvećim RSCU koji iznosi 1.8, a neka AAG ima 0.2. Tada je:

$$w = \frac{0.2}{1.8}$$

koliko je kodon dobar u odnosu na najbolji.

CAI je prosek svih w vrednosti. CAI blizak 1 govori o tome da gen koristi optimalne kodone i da ima visoku ekspresiju, a CAI blizak 0 govori da koristi loše kodone i da ima nisku ekspresiju. Ekspresija je uključenost gena u sintezu proteina. Na primer, gen za insulin ako se ekspresuje pravi se insulin, ako se ne ekspresuje ne pravi se insulin. CAI nam ustvari kaže koliko efikasno gen pravi protein. Ako je kodon "popularan" brzo se prevodi, ako nije, ribozom čeka i proces se usporava. Gen sa popularnijim kodonima ima ubrzan proces sinteze proteina i može da sintetiše veću količinu proteina.

Imamo i ugradjene funkcije koje nam izračunavaju pomenute metrike:

```
sequence = 'CGCATTACATTACGTACTGGTACA'
from CAI import RSCU, relative_adaptiveness, CAI
RSCU([sequence]) # RSCU iz CAI biblioteke ocekuje listu
    sekvenci kao argument
relative_adaptiveness([sequence])
```

1.1 Primer 1: Računanje RSCU nad kodirajućim sekvencama e. coli

FASTA fajlovi su jednostavan tekstualni format koji se koristi u bioinformatici za čuvanje bioloških sekvenci, kao što su DNK, RNK ili proteinske sekvence. Ovaj format je jedan od najčešće korišćenih standarda jer je lak za čitanje i obradu pomoću programskih jezika.

Svaka sekvenca u FASTA fajlu sastoji se iz dva osnovna dela:

- **Header (opis)** – linija koja počinje znakom `>` i sadrži identifikator sekvence i opcioni opis
- **Sekvenca** – niz karaktera koji predstavljaju nukleotide (A, T, C, G) ili aminokiseline

Primer FASTA zapisa:

```
>seq1 Homo sapiens gene
ATGCGTACGTAGCTAG
```

FASTA fajl može sadržati jednu ili više sekvenci, pri čemu svaka nova sekvenca počinje linijom sa znakom `>`.

Zbog svoje jednostavnosti i kompatibilnosti sa različitim bioinformatičkim alatima, FASTA format se široko koristi za:

- skladištenje sekvenci
- poredjenje sekvenci (npr. BLAST)
- analizu genoma i proteina

FASTA je osnovni i univerzalni format za rad sa biološkim sekvencama, koji omogućava lako čuvanje, razmenu i obradu podataka.

U narednom kodu najpre ćemo pročitati informacije o genima Ešerihije koli.

```
from Bio import SeqIO
import pandas as pd
records_raw = SeqIO.parse('data/e_coli_cds.fna', 'fasta')
sequences = [str(record.seq) for record in records_raw] # -
    Kopiranje radi mogucnosti kasnije ponovne iteracije
e_coli_rscu = RSCU(sequences)
e_coli_rscu_df = pd.DataFrame(data=[e_coli_rscu.values()],
    columns=e_coli_rscu.keys())
```

Ukoliko želimo malo složenije analize, možemo prvo nazive gena da povežemo sa njihovim zapisom:

```
'''Izdvajanje CDS kod e. coli koji odgovaraju imenovanim
    genima'''
import re # - Modul za rad sa regularnim izrazima

records_raw = SeqIO.parse('data/e_coli_cds.fna', 'fasta')
records = [record for record in records_raw] # - Kopiranje
    radi mogucnosti kasnije ponovne iteracije

gene_regex = r'gene=(.*?)\]' # - Regularni izraz za
    izdvajanje naziva gena iz FASTA zaglavlja
data = []
for record in records:
    match = re.search(gene_regex, record.description)
    if match != None:
        gene_name = match.groups()[0]
        sequence = str(record.seq)
        data.append([gene_name, sequence])
```

Od liste parova naziv gena - zapis gena, možemo lako napraviti tabelu na sledeći način:

```
gene_df = pd.DataFrame(data=data, columns=['gene', 'sequence
    '])
```

Ako hoćemo samo niz sekvenci, bez da znamo koje su možemo izvršiti sledeći kod:

```
gene_sequences = gene_df['sequence'].tolist()
```

A potom možemo saznati i RSCU nad svim sekvencama gena kod e. coli:

```
genome_rscu = RSCU(gene_sequences)
```

Zanima nas koji su geni najekspresivniji:

```
'''
Izracunavanje CAI pojedinacnih gena pri cemu se kao
    referentna vrednost koristi RSCU sekvenci svih gena
Napomena: Referentne RSCU vrednost treba racunati na osnovu
    sekvenci gena sa najvecom ekspresijom!
'''

cai_data = []

# Iteriranje kroz DataFrame gena - nukleotidnih sekvenci
    gena
for index, row in gene_df.iterrows():
    gene = row.gene
    sequence = row.sequence
    cai = CAI(sequence, RSCUs=genome_rscu) # - Racunanje CAI
    cai_data.append([gene, cai])

# Konstrukcija DataFrame-a sa nazivima gena i njihovim CAI
    vrednostima
cai_df = pd.DataFrame(data=cai_data, columns=['gene','CAI'])
cai_df
```

Računanje CAI vrednosti za gene

Ovaj kod računa **CAI (Codon Adaptation Index)** za svaki gen u datom skupu podataka. CAI predstavlja meru koliko je upotreba kodona u jednom genu slična upotrebi kodona u referentnom skupu gena, koji su najčešće visoko eksprimirani geni. Veće vrednosti CAI ukazuju na veću sličnost i potencijalno viši nivo ekspresije gena.

Opis algoritma

- Na početku se inicijalizuje prazna lista `cai_data`, koja će sadržati rezultate.
- Petlja:

  `for index, row in gene_df.iterrows():`

  prolazi kroz svaki red DataFrame-a `gene_df`, gde svaki red predstavlja jedan gen i njegovu nukleotidnu sekvencu.
- Iz svakog reda izdvajaju se:
  - `gene` – naziv gena

  - `sequence` – DNK sekvenca gena

- Za svaki gen se računa CAI:

  ```
  cai = CAI(sequence, RSCUs=genome_rscu)
  ```

  pri čemu se koriste **RSCU vrednosti referentnog skupa gena**. Ove vrednosti treba da potiču od visoko eksprimiranih gena, jer CAI meri prilagodjenost kodonske upotrebe tom referentnom skupu [**?**].

- Rezultat (naziv gena i njegova CAI vrednost) se dodaje u listu:

  ```
  cai_data.append([gene, cai])
  ```

- Nakon obrade svih gena, pravi se novi DataFrame:

  ```
  cai_df = pd.DataFrame(data=cai_data, columns=['gene','CAI'])
  ```

- Dobijeni DataFrame sadrži dve kolone:
  - naziv gena
  - odgovarajuću CAI vrednost

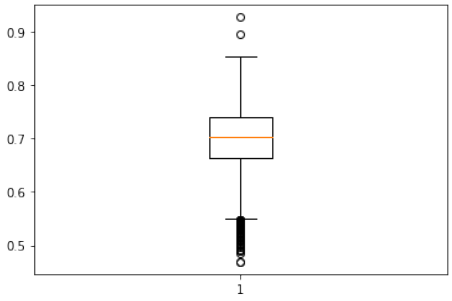
Možemo dobiti spisak najekspresivnijih gena sortiranjem dobijenih CAI vrednosti:

```
'''Generisanje sortiranog DataFrame-a sa CAI vrednostima
    gena u opadajucem redosledu'''

sorted_cai_df = cai_df.sort_values(by='CAI', ascending=False
    )
sorted_cai_df
```

Lako se može napraviti i dijagram na osnovu koga se može lako sagledati ceo skup vrednosti za CAI:

```
import matplotlib.pyplot as plt
_ = plt.boxplot(sorted_cai_df.iloc[:,1])
```

Većina gena (preko 50%) ima vrednosti CAI unutar pravougaonika na slici. Dosta gena ima nisku ekspresiju, a samo malo njih izuzetno visoku (tačkice na slici).



1.2 Primer 2: Konstrukcija filogenetskog stabla na osnovu RSCU

Stablo je konstruisano na osnovu RSCU vrednosti HBB gena kod 5 različitih organizama. Za konstrukciju stabla korišćen je algoritam za hijerarhijsko klasterovanje iz biblioteke scipy.

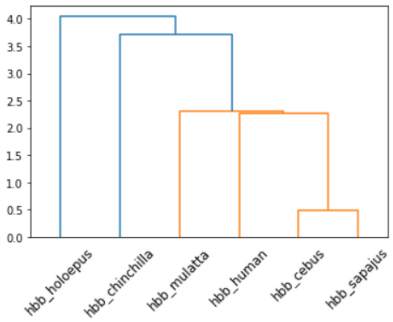
```
base = 'data/hbb_sekvence'
filepaths = [f'{base}/{f}' for f in os.listdir(base) if not
    f.startswith('.')]

data = []

for path in filepaths:
    record = SeqIO.read(path, 'fasta')
    label = path.split('/')[-1].split('.')[0]
    sequence = record.seq

    rscu_table = RSCU([sequence])

    row = {'label': label}
    row.update(rscu_table)

    data.append(row)

df = pd.DataFrame(data).reset_index(drop=True)
```

Dobićemo sledeću tabelu sa izračunatim RSCU kod HBB gena različitih organizama: Na osnovu dobijene tabele možemo uraditi hijerarhijsko klasterovanje

	label	TTT	TTC	TTA	TTG	TCT	TCC	TCA	TCG	TAT	...	GCA	GCG	GAT	GAC	
0	hbb_chinchilla	0.444444	1.555556	0.157895	0.157895	1.600000	1.600000	0.400000	0.400000	0.400000	...	0.108108	0.216216	0.500000	1.500000	0.54
1	hbb_human	1.250000	0.750000	0.150000	0.150000	0.923077	1.846154	0.461538	0.461538	1.333333	...	0.125000	0.125000	1.428571	0.571429	0.50
2	hbb_mulatta	1.250000	0.750000	0.153846	0.153846	0.923077	1.846154	0.461538	0.461538	0.666667	...	0.142857	0.142857	1.428571	0.571429	0.50
3	hbb_cebus	1.000000	1.000000	0.150000	0.150000	0.923077	1.846154	0.461538	0.461538	0.666667	...	0.133333	0.133333	1.000000	1.000000	0.85
4	hbb_sapajus	1.000000	1.000000	0.150000	0.150000	0.923077	1.846154	0.461538	0.461538	0.666667	...	0.133333	0.133333	1.000000	1.000000	0.85
5	hbb_holoepus	1.250000	0.750000	0.146341	0.878049	0.375000	3.000000	0.375000	0.375000	1.000000	...	0.133333	0.133333	1.111111	0.888889	1.25

i dendrogram.

```python
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# 1. Priprema podataka
# uklonimo label kolonu (to su imena gena)
X = df.drop(columns=['label']).values

# 2. Hijerarhijsko klasterovanje
Z = linkage(X, method='ward')  # moze i 'single', 'complete
    ', 'average'

# 3. Crtanje dendrograma
plt.figure(figsize=(10, 6))
dendrogram(Z, labels=df['label'].values)

plt.title("Dendrogram (Hijerarhijsko klasterovanje gena)")
plt.xlabel("Geni")
plt.ylabel("Distanca")

plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Hijerarhijsko klasterovanje i dendrogram

Hijerarhijsko klasterovanje je metoda nenadgledanog učenja koja grupiše objekte na osnovu njihove medjusobne sličnosti i formira hijerarhiju klastera. Za razliku od nekih drugih metoda, nije potrebno unapred zadati broj klastera, već se dobija kompletna struktura grupisanja podataka.

Postoje dva osnovna pristupa:

- **Aglomerativno (bottom-up)** – svaki objekat počinje kao poseban klaster, a zatim se najbliži klasteri postepeno spajaju
- **Divizivno (top-down)** – svi objekti počinju u jednom klasteru koji se zatim deli na manje

Rezultat hijerarhijskog klasterovanja se prikazuje pomoću **dendrograma**, koji predstavlja strukturu klastera u obliku stabla. Listovi stabla predstavljaju pojedinačne objekte, dok unutrašnji čvorovi predstavljaju spajanje klastera.

Ward metoda je posebna vrsta hijerarhijskog (aglomerativnog) klasterovanja koja ne gleda samo rastojanje izmedju klastera, već promenu varijanse unutar klastera.

Dendrogram

Dendrogram je dijagram koji prikazuje kako se objekti grupišu tokom klasterovanja. Visina na kojoj se dve grane spajaju predstavlja njihovu medjusobnu udaljenost (razliku).

Važne osobine dendrograma:

- listovi predstavljaju pojedinačne objekte (npr. gene)
- grane predstavljaju spajanje klastera
- visina spajanja označava koliko su objekti različiti

Dendrogram omogućava da se broj klastera odredi tako što se "preseče" na odredjenoj visini, pri čemu objekti ispod preseka pripadaju istom klasteru.

Primena na RSCU podatke

U ovom primeru, hijerarhijsko klasterovanje se primenjuje na gene opisane njihovim RSCU vrednostima. Svaki gen se posmatra kao vektor numeričkih vrednosti koje predstavljaju upotrebu kodona.

Klasterovanje omogućava:

- identifikaciju gena sa sličnim obrascima upotrebe kodona
- grupisanje gena koji mogu imati sličan nivo ekspresije
- otkrivanje potencijalnih funkcionalnih ili evolutivnih sličnosti

Na dendrogramu:

- geni koji se spajaju na maloj visini imaju sličnu kodonsku upotrebu
- geni koji se spajaju na većoj visini su medjusobno različitiji

Zaključak

Hijerarhijsko klasterovanje i dendrogram predstavljaju moćan alat za vizualizaciju i analizu sličnosti izmedju gena. U kontekstu RSCU vrednosti, oni omogućavaju bolje razumevanje obrazaca kodonske upotrebe i potencijalnih bioloških odnosa izmedju gena.

2 Probabilistički grafički modeli - uvod

Kompleksni sistemi su okarakterisani mnogim nekorelisanim aspektima od kojih mnogi utiču na ishod zadatka o kome rezonujemo. Na primer, u slučaju postavljanja dijagnoze, postoji mnogo bolesti koje pacijent može da ima, stotine ili hiljade simptoma i dijagnostičkih testova, lične karakteristike koje predstavljaju predispozicije za neke bolesti i još gomila faktora. Ovi domeni mogu se predstaviti slučajnim promenljivama (*random variables*). Konkretna bolest, kao što je Grip (*Flu*) može imati vrednosti prisutan (*present*) ili odsutan (*absent*), simptom ove bolesti Groznica (*Fever*) može imati neprekidnu vrednost. Skup promenljivih i njihovih vrednosti važno je pitanje dizajna i zavisi od toga na kakvo pitanje o problemu želimo da damo odgovor.

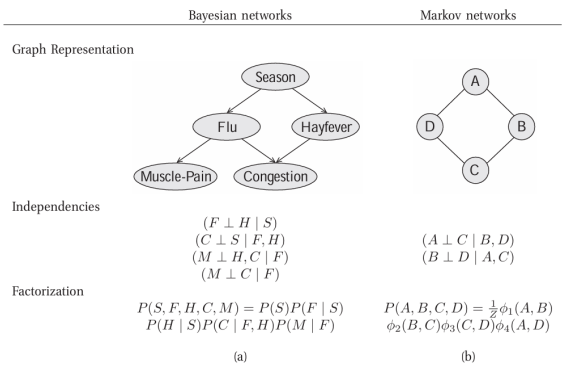
Naš zadatak je da na probabilistički način rezonujemo o vrednosti jedne ili više promenljivih, ako su nam pritom date observacije o drugim promenljivama. Da bismo to uradili, konstruišemo zajedničku raspodelu (*joint distribution*) nad skupom slučajnih promenljivih X. Ovaj tip modela nam omogućuje da odgovorimo na veliki skup interesantnih upita. Na primer, možemo da napravimo pretpostavku da slučajna promenljiva X_i uzima vrednost x_i i da se pitamo kakva je raspodela verovatnoća nad vrednostima druge promenljive X_j.

*Razmotrimo jedan veoma jednostavan primer. Posmatramo Grip i Prehladu. Ove dve bolesti nisu uzajamno isključive, neko može imati i jedno i drugo. Takodje imamo jednu slučajnu veličinu Sezona, koja može da uzima 4 vrednosti - Proleće, Leto, Jesen i Zima. I dva simptoma BolMišića i ZačepljenNos, za koje beležimo da li su prisutni ili ne. Ukupno imamo 2*2*4*2*2 = 64 moguće kombinacije. Ako nam je data zajednička raspodela, možemo da se pitamo kolika je verovatnoća da pacijent ima Grip ako je Jesen i ima ZačepljenNos, ali ne i BolMišića. Ovo se matematički može zapisati kao:*

$$P(Flu = true)|(Seasson = fall, Congestion = true, MusclePain = false)$$

2.1 Probabilistički grafički modeli

Probabilistički grafički modeli služe da nam omoguće da kompaktno zapišemo komplikovane raspodele u visoko dimenzionim prostorima. Čvorovi na graficima odgovaraju promenljivama, a grane direktnim probabilističkim vezama izmedju njih: Za primer koji smo malopre imali vidimo da nema interakcije izmedju bola u mišićima i sezone, ali oba direktno interaguju sa gripom. Ovo je dualna perspektiva koju možemo iskoristiti da interpretiramo graf. Oznaka $(X \perp Y|Z)$ znači da je X nezavisno od Y ako je vrednost od Z poznata. Na primer, (ZačepljenNos $\perp$ Sezona | Grip, Prehlada) znači da ako je poznato da li pacijent ima grip i da li ima prehladu onda sezona nije relevantna informacija za to da li mu je nos začepljen ili ne. Ipak moramo paziti, ovo ne znači da sezona ne utiče na raspodelu verovatnoća za začepljenost nosa, već da sve informacije koje je sezona proizvela i sav uticaj se preslikao kroz podatke o gripu i prehladi. Drugi važan razlog zašto nam je graf potreban je što nam pomaže da neku komplikovanu verovatnoću sa puno promenljivih razbijemo na faktore koje množimo i koji su



jednostavni. Recimo verovatnoća dogadjaja: "proleće je, osoba nema grip, ima prehladu, začepljen nos i bol u mišićima" može se zapisati kao proizvod sledećih verovatnoća:

$$P(Season = spring), P(Flu = false|Season = spring), P(Hayfever = true|Season = spring)$$

$$P(Congestion = true|Flu = false, Hayfever = true), P(MusclePain = true|Flu = False)$$

U ovim izrazima jasno je da verovatnoća neke konkretne vrednosti zavisi samo od čvorova koji su u grafu spojeni sa čvorom te promenljive. Izučavaćemo dve familije grafičkog prikaza distribucija: Bajesove mreže - koje su usmereni grafovi i Markovljeve - koje su neusmereni. Sa obe čitamo nezavisnosti i faktorizacije.

2.2 Bajesove mreže

Naš cilj je da predstavimo zajedničku raspodelu P nad nekim skupom slučajnih promenljivih $X = \{X_1, ..., X_n\}$. Čak i u najjednostavnijem slučaju kada su sve promenljive binarne, zajednička raspodela bi zahtevala verovatnoće 2^n različitih mogućnosti. Eksplicitno čuvanje takvih informacija je nezamislivo iz svake perspektive. Računski je veoma skupo čuvati i manipulisati sa tom količinom informacija, kognitivno je nemoguće prikupiti toliko informacija od ljudskih eksperata, statistički ako hoćemo raspodelu da naučimo iz podataka, užasno mnogo podataka bi nam trebalo da naučimo parametre robusno. Svi ovi razlozi doveli su do razvoja teorije o kojoj ćemo ovde da govorimo.

Razmotrimo sledeći primer. Kompanija želi da zaposli pametne studente, ali nema način da direktno testira inteligenciju. Jedino što poseduje je pristup SAT skoru, koji jeste informativan ali ne potpuno indikativan. Znači naš prostor verovatnoća indukovan je dvema važnim promenljivama Inteligencija i SAT. Zbog jednostavnosti možemo pretpostaviti da svaka od njih uzima po dve vrednosti: $Val(I) = \{i^1, i^0\}$ *i* $Val(S) = \{s^1, s^0\}$ *pri čemu je* 1 *oznaka za višu vrednost i* 0 *za nižu. Primer jedne zajedničke raspodele može biti:*

i^0	s^0	*0.665*
i^0	s^1	*0.035*
i^1	s^0	*0.06*
i^1	s^1	*0.24*

Postoji naravno alternativa. Možemo zapisati: $P(I,S) = P(I) * P(S|I)$. *Predstavljamo proces na način koji je intuitivno saglasniji sa uslovnošću. Gomila faktora (geni, okruženje, dostupnost resursa,...) uticali su najpre na razvoj studentove inteligencije. Njegov SAT skor je potom pod uticajem inteligencije. Iz matematičke perspektive, ovakav način razmišljanja nas dovodi do toga da umesto da čuvamo jednu visoko dimenzionu tabelu, čuvamo nekoliko manjih, a vrednosti verovatnoća koje nas zanimaju dobijamo množenjem odgovarajućih faktora. Za ovaj konkretni slučaj čuvamo* $P(I)$*:*

i^0	*0.7*
i^1	*0.3*

i $P(S|I)$

I	s^0	s^1
i^0	*0.95*	*0.05*
i^1	*0.2*	*0.8*

Verovatnoća da student dobije veliki SAT skor, a da nije visoko inteligentan je mala 0.05*, sa druge strane verovatnoća da dobije veliki SAT skor ako je visoko inteligentan je velika* 0.8*, ali nije* 100%*.*

Pretpostavimo sada da osim ova dva, imamo i ocenu (prosečnu, ali jednostavnosti radi smatrajmo je diskretnom) studenta koja može uzeti neke tri vrednosti npr. $Val(G) = \{g^1, g^2, g^3\}$ *koje predstavljaju ocene A,B i C. Inteligencija studenta je očigledno povezana sa SAT skorom i sa ocenom. Takodje i SAT skor i ocena su povezani, ali su uslovno nezavisni u smislu koji smo već opisali, tj. ako znamo inteligenciju studenta, skor nam ne nosi nikakvu novu informaciju o oceni. Odnosno, formalnije zapisano:*

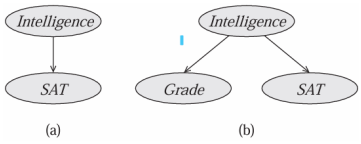


Figure 3.1 Simple Bayesian networks for the student **example**

$$P(g|i^1, s^1) = P(g|i^1)$$

ili

$$S \perp G|I$$

Ovde je bitna pretpostavka da je inteligencija studenta jedini razlog, jedina veza izmedju SAT skora i ocene. Naravno da ovo nije realna pretpostavka, ali

je možda dobra aproksimacija sistema. Ovde se već može naslutiti intuicija sa grafika, za sve informacije koje su poznate, obrišemo njihove čvorove, oni čvorovi koji ostanu nepovezani u takvom grafiku uslovno su nezavisni. Možemo faktorisati raspodelu verovatnoća:

$$P(I, S, G) = P(S, G|I) * P(I)$$

$$P(S, G|I) = P(S|I) * P(G|I)$$

jer I utiče na S i G, a S i G su uslovno nezavisne u odnosu na I, odakle je konačno:

$$P(I, S, G) = P(S|I) * P(G|I) * P(I)$$

2.2.1 Naivni Bajes

U ovoj podsekciji opisaćemo naivni Bajesov model (*Idiot Bayes model*). Pretpostavka je da postoji jedna promenljiva C (u primeru sa studentima je to Inteligencija) koja utiče na ostale $\{X_1, ..., X_n\}$ (SAT skor, ocena) i jedina je stvar koja utiče, odnosno one su sve medjusobno uslovno nezavisne - kad je C poznato, one ne zavise jedna od druge. Raspodela verovatnoća se ovde faktoriše

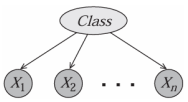


Figure 3.2 The Bayesian network graph for a naive Bayes model

kao:

$$P(C, X_1, ..., X_n) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

Ovi faktori se mogu sačuvati sa malim brojem parametara - linearnim, za razliku od eksponencijalnog broja kada bismo čuvali zajedničku raspodelu.

Bez obzira što pravi ozbiljno jake pretpostavke, Bajesov model se u praksi koristi zbog svoje jednostavnosti i malog broja potrebnog parametra. Značajna je primena za klasifikaciju - odlučivanje zasnovano na vrednosti nekih parametara koji predstavljaju dokaze za datu instancu, kojoj klasi pripada. Formalno, to možemo zapisati kao:

$$\frac{P(C = c^1|x_1, ..., x_n)}{P(C = c^2|x_1, ..., x_n)} = \frac{P(C = c^1)}{P(C = c^2)} \prod_{i=1}^{n} \frac{P(x_i|C = c^1)}{P(x_i|C = c^2)}$$

Interesantno je da se u svojim ranim danima, ovaj model koristio za medicinske dijagnoze, promenljiva C je predstavljala bolest (njene vrednosti su odgovarale različitim bolestima) a promenljive $X_1, ..., X_n$ simptome, i rezultate različitih testova. Ovaj model je korišćen za medicinsku dijagnostiku jer je mali broj interpretabilnih parametara olakšavao njihovo dobijanje od stručnjaka. Na

primer, sasvim je prirodno pitati lekara eksperta kolika je verovatnoća da pacijent sa upalom pluća ima visoku temperaturu. Zaista, nekoliko ranih sistema za medicinsku dijagnostiku bilo je zasnovano na ovoj tehnologiji, a pokazano je da su neki od njih davali bolje dijagnoze od onih koje postavljaju lekari eksperti. Naravno, ovaj metod ima i ozbiljne mane. Pretpostavlja da pacijent nema istovremeno više bolesti, a takodje i da su $X_1, ..., X_n$ medjusobno nezavisni, a oni mogu biti baš ozbiljno korelisani, na primer gojaznost i visok pritisak su oba znaci za probleme sa srcem, ali pošto su povezani, njihov uticaj se duplo računa. Iz ovog razloga razvijeni su kompleksniji Bajesovi modeli. Ipak, naivni Bajes se tradicionalno koristi u oblasti gde daje dobre rezultate, domenima sa puno featura i malo x-eva, kao što je recimo klasifikacija dokumenta u teme, na osnovu reči koje se javljaju u tim dokumentima.

2.3 Bajesove mreže

U ovom modelu posmatra se nešto složenija situacija u kojoj ocena studenta ne zavisi samo od njegove inteligencije, već i od težine kursa. Pored toga, profesor piše preporuku za studenta, ali pošto ga se ne seća, oslanja se isključivo na njegovu ocenu, pa kvalitet preporuke zavisi od ocene i odredjen je nasumično. U modelu se zato pojavljuje pet slučajnih promenljivih: inteligencija (I), težina kursa (D), ocena (G), SAT rezultat (S) i kvalitet preporuke (L), pri čemu su sve binarne osim ocene koja ima tri vrednosti, pa ukupna zajednička raspodela ima 48 mogućih kombinacija .

Ovaj sistem se predstavlja Bajesovom mrežom, odnosno usmerenim grafom u kome čvorovi predstavljaju promenljive, a grane direktne uticaje medju njima. Ideja je da se ceo proces može posmatrati kao generativni proces u kome svaka promenljiva dobija vrednost na osnovu verovatnoće koja zavisi samo od njenih "roditelja". Tako inteligencija i težina kursa nezavisno utiču na ocenu, inteligencija dodatno utiče na SAT rezultat, dok kvalitet preporuke zavisi samo od ocene. Na taj način model jasno prikazuje lokalne zavisnosti izmedju promenljivih, koje zajedno formiraju celokupno ponašanje sistema.

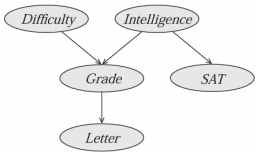
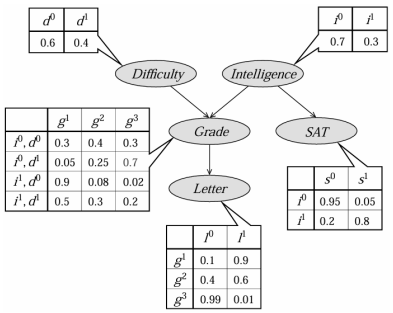


Figure 3.3 The Bayesian Network graph for the Student **example**

Moramo imati i uslovne verovatnoće kao na slici: da bi mogli da izračunamo npr. verovatnoću da inteligentan student na lakom kursu ima nisku ocenu a da je imao visok SAT skor i loše pismo:

$$P(i^1, d^0, g^2, s^1, l^0) = P(i^1)P(d^0)P(g^2|i^1, d^0)P(s^1|i^1)P(l^0|g^2) = 0.004608$$



Bajesova mreža predstavlja zajedničku raspodelu verovatnoća, iz koje možemo računati bilo koju uslovnu verovatnoću $P(Y|E)$. To radimo tako što uzmemo samo one slučajeve koji su u skladu sa posmatranim dokazima (evidence). Tipovi zaključivanja koje možemo da sprovedemo u ovakvim mrežama su sledeći:

- kauzalno zaključivanje (uzrok → posledica)
  - na primer inteligencija → ocena → preporuka
  - ako znamo da je student slab, smanjuje se verovatnoća dobre preporuke

  ovo je predikcija
- evidencijalno zaključivanje (posledica → uzrok)
  - loša ocena → manja verovatnoća da je student visoko inteligentan
  - preporuka takodje daje informaciju ali slabiju od ocene

  ovo je objašnjenje
- interkauzalno zaključivanje (izmedju uzroka) - dešava se kada više uzroka utiče na istu posledicu, najvažniji efekat je *explaining away*:
  - ako znamo posledicu npr. loša ocena, moguće je više uzroka (niska inteligencija ili težak kurs)
  - ako otkrijemo jedan uzrok - npr. kurs je težak, verovatnoća drugog uzroka se menja - u ovom slučaju smanjuje se verovatnoća niske inteligencije

2.4 Bajesovi modeli i osnovne komponente u `pgmpy`

Bajesovi modeli predstavljaju jedan od najznačajnijih pristupa u okviru probabilističkih grafičkih modela. Oni omogućavaju modelovanje zavisnosti izmedju slučajnih promenljivih na formalan i strukturiran način, koristeći usmerene aciklične grafove. U kontekstu bioinformatike, ovakvi modeli se često koriste

za opisivanje složenih bioloških procesa, kao što su regulacija ekspresije gena, signalni putevi ili odnosi izmedju genetičkih i fenotipskih karakteristika.

Implementacija Bajesovih modela u programskom jeziku Python može se realizovati pomoću biblioteke `pgmpy`, koja pruža alate za definisanje strukture modela, zadavanje verovatnoća i izvodjenje zaključivanja. Osnovne komponente koje ćemo koristiti date su u nastavku:

Listing 1: Osnovne komponente za rad sa Bajesovim modelom

```
# Bajesov model
from pgmpy.models import BayesianModel

# Conditional Probability Table, funkcije uslovne raspodele
    tabelarno predstavljena
from pgmpy.factors.discrete import TabularCPD

# Algoritam koji vrsi zakljucivanje
from pgmpy.inference import VariableElimination
```

Klasa `BayesianModel` koristi se za definisanje strukture modela, odnosno zavisnosti izmedju promenljivih. Klasa `TabularCPD` omogućava zadavanje uslovnih raspodela verovatnoća u tabelarnom obliku, što predstavlja ključni korak u parametrizaciji modela. Na kraju, `VariableElimination` predstavlja algoritam za zaključivanje, koji omogućava izračunavanje verovatnoća interesantnih dogadjaja na osnovu poznatih podataka.

U nastavku ćemo, polazeći od ovih osnovnih komponenti, konstruisati konkretan Bajesov model i analizirati njegovo ponašanje.

2.4.1 Algoritam eliminacije promenljivih

Jedan od osnovnih zadataka u radu sa Bajesovim mrežama jeste **zaključivanje**, odnosno izračunavanje verovatnoća odredjenih promenljivih na osnovu poznatih informacija. U tu svrhu koristi se više algoritama, a jedan od najpoznatijih i najčešće korišćenih jeste **eliminacija promenljivih** (*Variable Elimination*).

Algoritam eliminacije promenljivih zasniva se na ideji da se zajednička raspodela verovatnoće može efikasno izračunati tako što se postepeno eliminišu (sumiraju) promenljive koje nisu od interesa. Umesto da se računa kompletna zajednička raspodela nad svim promenljivima, što je u praksi često neizvodljivo zbog eksponencijalnog rasta, ovaj algoritam koristi strukturu grafa kako bi značajno smanjio broj operacija.

Intuitivno, postupak funkcioniše tako što:

- identifikujemo promenljive koje želimo da izračunamo (upit),
- uzimamo u obzir poznate vrednosti nekih promenljivih (evidenciju),
- ostale promenljive postepeno eliminišemo sumiranjem preko njihovih mogućih vrednosti.

Tokom ovog procesa, koriste se lokalne funkcije verovatnoće (CPD-ovi), koje se medjusobno množe i marginalizuju. Redosled eliminacije promenljivih može značajno uticati na efikasnost algoritma, jer odredjuje veličinu medjukoraka (intermedijarnih tabela).

U biblioteci `pgmpy`, algoritam eliminacije promenljivih implementiran je kroz klasu `VariableElimination`. Nakon što definišemo model i njegove raspodele verovatnoća, možemo inicijalizovati objekat za zaključivanje na sledeći način:

Listing 2: Inicijalizacija algoritma eliminacije promenljivih

```
from pgmpy.inference import VariableElimination

inference = VariableElimination(model)
```

Nakon inicijalizacije, moguće je postavljati upite nad modelom. Na primer, možemo izračunati raspodelu verovatnoće za odredjenu promenljivu ili skup promenljivih, uz eventualno zadatu evidenciju. Ovakav pristup omogućava fleksibilno i efikasno zaključivanje u složenim probabilističkim modelima.

Važno je naglasiti da eliminacija promenljivih daje **tačne rezultate**, za razliku od nekih aproksimativnih metoda, ali njena vremenska i memorijska složenost može biti velika kod mreža sa velikim brojem čvorova ili gustim povezivanjem. Uprkos tome, zbog svoje konceptualne jednostavnosti i preciznosti, ovaj algoritam predstavlja standardni alat za zaključivanje u Bajesovim mrežama.

Marginalna verovatnoća Marginalna verovatnoća predstavlja verovatnoću jedne slučajne promenljive, nezavisno od vrednosti ostalih promenljivih u modelu. Drugim rečima, ona se dobija tako što se "zanemare" (eliminišu) ostale promenljive, najčešće sumiranjem preko svih njihovih mogućih vrednosti.

Neka imamo dve slučajne promenljive X i Y sa zajedničkom raspodelom $P(X, Y)$. Marginalna verovatnoća promenljive X definiše se kao:

$$P(X) = \sum_{Y} P(X, Y)$$

Intuitivno, ovo znači da verovatnoću dogadjaja X dobijamo tako što sabiramo verovatnoće svih mogućih slučajeva u kojima se X pojavljuje, bez obzira na to koja je vrednost promenljive Y.

Na primer, ako promenljiva Y može da uzme dve vrednosti y_1 i y_2, tada važi:

$$P(X) = P(X, y_1) + P(X, y_2)$$

U kontekstu Bajesovih mreža, marginalna verovatnoća se često računa pomoću algoritma eliminacije promenljivih, gde se neželjene promenljive eliminišu upravo ovakvim sumiranjem. Na taj način dobijamo verovatnoću promenljive od interesa bez potrebe da eksplicitno razmatramo sve kombinacije vrednosti u modelu.

Primer rada algoritma eliminacije promenljivih Da bismo razumeli kako funkcioniše eliminacija promenljivih, razmotrimo jednostavan model sa tri promenljive:

- A – uzrok,
- B – posredna promenljiva,
- C – posledica.

Pretpostavimo da važi struktura:

$$A \to B \to C$$

Zajednička raspodela ovih promenljivih može se zapisati kao:

$$P(A, B, C) = P(A) \cdot P(B \mid A) \cdot P(C \mid B)$$

Neka nas zanima marginalna verovatnoća promenljive C, odnosno $P(C)$. Direktno računanje bi zahtevalo sumiranje preko svih mogućih vrednosti promenljivih A i B:

$$P(C) = \sum_A \sum_B P(A) \cdot P(B \mid A) \cdot P(C \mid B)$$

Algoritam eliminacije promenljivih omogućava da se ovaj izraz izračuna efikasnije, reorganizacijom izraza i postepenim eliminisanjem promenljivih.

Prvo eliminišemo promenljivu A. Grupisanjem odgovarajućih faktora dobijamo:

$$\sum_A P(A) \cdot P(B \mid A)$$

Rezultat ove sume je nova funkcija koja zavisi samo od promenljive B (to je zapravo marginalna raspodela $P(B)$). Označimo je sa $f(B)$:

$$f(B) = \sum_A P(A) \cdot P(B \mid A)$$

Sada izraz za $P(C)$ postaje:

$$P(C) = \sum_B f(B) \cdot P(C \mid B)$$

U sledećem koraku eliminišemo promenljivu B:

$$P(C) = \sum_B \left(\sum_A P(A) \cdot P(B \mid A) \right) \cdot P(C \mid B)$$

Na ovaj način izbegavamo računanje kompletne zajedničke raspodele $P(A, B, C)$, već radimo sa manjim, lokalnim funkcijama.

Ključna ideja algoritma je sledeća:

- kombinujemo (množimo) samo relevantne faktore,

- eliminišemo promenljive sumiranjem,
- kreiramo nove, manje faktore koji se koriste u narednim koracima.

Ovakav pristup značajno smanjuje broj operacija u odnosu na naivno računanje, posebno kod modela sa većim brojem promenljivih. Upravo zbog toga, eliminacija promenljivih predstavlja osnovni algoritam za tačno zaključivanje u Bajesovim mrežama.

Numerički primer marginalne verovatnoće i eliminacije promenljivih
Razmotrimo model sa tri binarne promenljive A, B i C:

$$A \to B \to C$$

Neka su date sledeće verovatnoće:

- Apriorna verovatnoća:

$$P(A = 1) = 0.3, \quad P(A = 0) = 0.7$$

- Uslovna verovatnoća B dato A:

$$P(B = 1 \mid A = 1) = 0.8, \quad P(B = 0 \mid A = 1) = 0.2$$
$$P(B = 1 \mid A = 0) = 0.2, \quad P(B = 0 \mid A = 0) = 0.8$$

- Uslovna verovatnoća C dato B:

$$P(C = 1 \mid B = 1) = 0.9, \quad P(C = 0 \mid B = 1) = 0.1$$
$$P(C = 1 \mid B = 0) = 0.1, \quad P(C = 0 \mid B = 0) = 0.9$$

Želimo da izračunamo marginalnu verovatnoću $P(C = 1)$.

Korak 1: eliminacija promenljive A
Prvo računamo raspodelu promenljive B:

$$P(B = 1) = P(A = 1)P(B = 1 \mid A = 1) + P(A = 0)P(B = 1 \mid A = 0)$$
$$P(B = 1) = 0.3 \cdot 0.8 + 0.7 \cdot 0.2 = 0.24 + 0.14 = 0.38$$

$$P(B = 0) = 0.3 \cdot 0.2 + 0.7 \cdot 0.8 = 0.06 + 0.56 = 0.62$$

Korak 2: eliminacija promenljive B
Sada računamo $P(C = 1)$:

$$P(C = 1) = P(B = 1)P(C = 1 \mid B = 1) + P(B = 0)P(C = 1 \mid B = 0)$$
$$P(C = 1) = 0.38 \cdot 0.9 + 0.62 \cdot 0.1 = 0.342 + 0.062 = 0.404$$

Dakle, dobijamo:

$$P(C = 1) = 0.404$$

Ovaj primer jasno ilustruje ideju eliminacije promenljivih: prvo smo eliminisali A da bismo dobili raspodelu nad B, a zatim eliminisali B da bismo dobili verovatnoću za C. Na taj način izbegavamo računanje kompletne zajedničke raspodele nad svim promenljivima.

Primer: Bajesova mreža za dijagnostiku COVID-19 Razmotrimo jednostavan model koji opisuje odnos izmedju simptoma i dijagnostičkih nalaza kod bolesti COVID-19. Posmatramo sledeće promenljive:

- `kasalj` – da li pacijent ima kašalj,
- `temperatura` – da li pacijent ima povišenu temperaturu,
- `X-ray` – rezultat rendgenskog snimka,
- `covid` – da li je pacijent zaražen.

Struktura modela je takva da simptomi utiču na rezultat rendgenskog snimka, dok nalaz rendgena dalje utiče na procenu. Na osnovu ove strukture, definišemo odgovarajuće uslovne raspodele verovatnoća (CPD-ove):

Listing 3: Definisanje CPD-ova za model COVID dijagnostike

```
# Kasalj, Temperatura, X-ray, COVID

'''
kasalj    temperatura
     \    /
      X-ray
        |
      covid
'''

K_cpd = TabularCPD(
    variable="kasalj",
    variable_card=2,
    values=[[0.4],[0.6]],
    state_names = { 'kasalj': ['NE', 'DA'] }
)

T_cpd = TabularCPD(
    variable="temperatura",
    variable_card=2,
    values=[[0.3],[0.7]],
    state_names = { 'temperatura': ['N', 'V'] }
)

X_cpd = TabularCPD(
    variable="X-ray",
    variable_card=2,
    evidence=['kasalj','temperatura'],
    evidence_card=[2,2],
    values=[
        [0.1,0.2,0.3,0.4],
        [0.9,0.8,0.7,0.6],
    ],
```

```
    state_names = {
        'kasalj': ['NE', 'DA'],
        'temperatura': ['N', 'V'],
        'X-ray': ['LOS', 'DOBAR']
    }
)

C_cpd = TabularCPD(
    variable="covid",
    variable_card=2,
    evidence=['X-ray'],
    evidence_card=[2],
    values=[
        [0.2,0.7],
        [0.8,0.3],
    ],
    state_names = {
        'X-ray': ['LOS', 'DOBAR'],
        'covid': ['NE', 'DA']
    }
)
```

U ovom modelu, promenljive `kasalj` i `temperatura` imaju apriorne raspodele, dok promenljiva `X-ray` zavisi od obe. Na kraju, promenljiva `covid` zavisi isključivo od rezultata rendgenskog snimka.

Ovakav model omogućava da, na osnovu simptoma i dijagnostičkih nalaza, procenimo verovatnoću prisustva bolesti, kao i da analiziramo kako različiti faktori utiču na konačnu dijagnozu.

Konstrukcija modela Nakon definisanja uslovnih raspodela verovatnoća (CPD-ova), sledeći korak je konstrukcija same Bejzesovske mreže, odnosno definisanje strukture grafa i povezivanje sa odgovarajućim verovatnoćama.

Struktura modela se zadaje navodjenjem usmerenih grana izmedju promenljivih, čime se eksplicitno definišu njihove zavisnosti. U našem primeru, promenljive `kasalj` i `temperatura` utiču na `X-ray`, dok `X-ray` dalje utiče na promenljivu `covid`.

Listing 4: Konstrukcija Bajesovog modela

```
# Konstrukcija modela pomocu CPD i veza izmedju cvorova
model = BayesianModel([
    ('kasalj', 'X-ray'),
    ('temperatura', 'X-ray'),
    ('X-ray', 'covid')
])

model.add_cpds(K_cpd, T_cpd, X_cpd, C_cpd)
```

Pozivom funkcije `BayesianModel` definišemo strukturu grafa, dok metod `add_cpds` služi za pridruživanje prethodno definisanih raspodela verovatnoća

svakom čvoru u modelu.

Na ovaj način dobijamo kompletno definisan Bajesov model, koji se sastoji iz dva ključna dela: strukture (graf zavisnosti) i parametara (verovatnoće). Takav model je spreman za dalje analize, uključujući izvodjenje zaključivanja pomoću algoritama kao što je eliminacija promenljivih.

Provera ispravnosti modela Nakon definisanja strukture mreže i dodavanja svih uslovnih raspodela verovatnoća, potrebno je proveriti da li je model ispravno konstruisan. U biblioteci `pgmpy` to se radi pomoću metode `check_model()`.

Listing 5: Provera ispravnosti modela

```
# Provera ispravnosti modela (probati sa izmenama vrednosti
    CPD)
model.check_model()
```

Ova metoda proverava da li su svi čvorovi u modelu dobili odgovarajuće CPD-ove, da li su roditelji navedeni u tabelama verovatnoća uskladjeni sa granama u grafu, kao i da li su same tabele korektno definisane. Posebno je važno da verovatnoće u svakoj koloni uslovne tabele daju zbir jednak 1, jer svaka kolona predstavlja raspodelu verovatnoće za fiksirane vrednosti roditeljskih promenljivih.

Ako je model ispravno definisan, metoda vraća vrednost `True`. U suprotnom, javlja se greška, što može pomoći da se uoče problemi u definiciji modela, na primer pogrešno zadate verovatnoće, neuskladjenost izmedju strukture mreže i CPD-ova ili izostavljanje raspodele za neki čvor.

Zbog toga je `check_model()` koristan korak u radu, naročito tokom testiranja i izmene modela, jer omogućava da se greške otkriju pre nego što se pristupi zaključivanju.

Izvršavanje probabilističkih upita Kada je model konstruisan i provereno da je ispravan, možemo pristupiti zaključivanju. U biblioteci `pgmpy` za to se koristi klasa `VariableElimination`, koja implementira algoritam eliminacije promenljivih. Ovaj algoritam omogućava izračunavanje verovatnoća promenljivih od interesa na osnovu strukture modela i definisanih uslovnih raspodela.

Listing 6: Izvršavanje probabilističkog upita nad modelom

```
# Izvrsavanje probabilistickih upita
inference = VariableElimination(model)
res = inference.query(['covid'])
print(res)
```

U prvoj liniji formira se objekat `inference`, koji predstavlja mehanizam za zaključivanje nad prethodno definisanim modelom. Drugom linijom zadaje se upit kojim se traži raspodela verovatnoće promenljive `covid`. Pošto u ovom slučaju nije navedena nikakva dodatna evidencija, računa se **marginalna verovatnoća** promenljive `covid`, odnosno njena verovatnoća bez uslovljavanja poznatim vrednostima drugih promenljivih.

Drugim rečima, model sabira doprinose svih mogućih kombinacija vrednosti promenljivih `kasalj`, `temperatura` i `X-ray`, kako bi dobio ukupnu raspodelu za promenljivu `covid`. Rezultat se smešta u promenljivu `res`, a zatim se ispisuje pomoću funkcije `print`.

Dobijeni rezultat predstavlja verovatnoće za obe moguće vrednosti promenljive `covid`, na primer za stanja `NE` i `DA`. Na taj način možemo videti kolika je ukupna verovatnoća prisustva, odnosno odsustva bolesti u okviru zadatog modela.

```
Finding Elimination Order: : 100%|██████████| 3/3 [00:00<00:00, 634.92it/s]
Eliminating: temperatura: 100%|██████████| 3/3 [00:00<00:00, 350.26it/s]
+-----------+--------------+
| covid     |   phi(covid) |
+===========+==============+
| covid(NE) |       0.5550 |
+-----------+--------------+
| covid(DA) |       0.4450 |
+-----------+--------------+
```

Izvršavanje upita uz zadatu evidenciju Pored izračunavanja marginalnih verovatnoća, Bejzesovska mreža omogućava i zaključivanje u situacijama kada su neke vrednosti promenljivih već poznate. Takve poznate vrednosti nazivaju se **evidencija** ili **opažanja**. Na osnovu njih moguće je izračunati uslovnu verovatnoću promenljive koja nas zanima.

Listing 7: Izvršavanje upita uz zadatu evidenciju

```
# Izvrsavanje upita uz opazanje pojedinih vrednosti
inference = VariableElimination(model)
res = inference.query(['covid'], evidence={'kasalj': 'DA', '
    temperatura': 'V'})
print(res)
```

U ovom primeru formira se objekat za zaključivanje nad modelom, a zatim se postavlja upit za promenljivu `covid`, pri čemu je zadato da pacijent ima kašalj i povišenu temperaturu, odnosno da važi `kasalj = DA` i `temperatura = V`. Na taj način ne računa se više ukupna, marginalna verovatnoća promenljive `covid`, već njena **uslovna verovatnoća** u prisustvu konkretnih opaženih simptoma.

Preciznije, izračunava se raspodela:

$$P(\texttt{covid} \mid \texttt{kasalj} = \texttt{DA},\ \texttt{temperatura} = \texttt{V})$$

To znači da model uzima u obzir samo one delove raspodele koji su u skladu sa zadatim vrednostima promenljivih `kasalj` i `temperatura`, a zatim na osnovu njih procenjuje verovatnoću da promenljiva `covid` ima vrednost `DA` ili `NE`. U tom procesu koristi se struktura mreže i odgovarajuće uslovne raspodele verovatnoća, posebno preko promenljive `X-ray`, koja povezuje simptome sa konačnom dijagnozom.

Ovakvi upiti su posebno značajni u bioinformatici i medicinskim primenama, jer omogućavaju procenu verovatnoće odredjenog stanja na osnovu već poznatih simptoma, nalaza ili eksperimentalnih opažanja. Na taj način Bajesove mreže

predstavljaju koristan alat za modelovanje neizvesnosti i donošenje zaključaka u složenim sistemima.

Primer uslovne nezavisnosti i zatvaranja toka u grafu Jedna od ključnih osobina Bajesovih mreža jeste mogućnost uočavanja **uslovne nezavisnosti** izmedju promenljivih na osnovu strukture grafa. Ova osobina se često objašnjava kroz pojam "zatvaranja toka informacija" u grafu.

Listing 8: Primer uslovne nezavisnosti u modelu

```
# Primer nezavisnosti usled zatvaranja toka zavisnosti u
    grafu
inference = VariableElimination(model)

res1 = inference.query(['covid'], evidence={'X-ray': 'LOS'})
print(res1)

res2 = inference.query(['covid'], evidence={'X-ray': 'LOS',
    'kasalj': 'NE'})
print(res2)
```

U ovom primeru posmatramo kako dodatna informacija o promenljivoj `kasalj` utiče na verovatnoću promenljive `covid`, u situaciji kada već znamo vrednost promenljive `X-ray`.

Kada zadamo evidenciju `X-ray` = `LOS`, čvor `X-ray` "zatvara" tok informacija izmedju njegovih roditelja (`kasalj`, `temperatura`) i potomka (`covid`). To znači da dodatna informacija o promenljivoj `kasalj` više ne utiče na verovatnoću promenljive `covid`.

Drugim rečima, važi približno:

$$P(\texttt{covid} \mid \texttt{X-ray}) = P(\texttt{covid} \mid \texttt{X-ray}, \texttt{kasalj})$$

što znači da su promenljive `covid` i `kasalj` **uslovno nezavisne** kada je poznata vrednost promenljive `X-ray`.

Ovo se može proveriti u praksi poredjenjem rezultata `res1` i `res2`, koji će biti jednaki (ili vrlo bliski), što potvrdjuje teorijsko svojstvo modela.

Ovaj fenomen je poznat kao **d-separacija** i predstavlja osnovni mehanizam pomoću kojeg struktura grafa odredjuje koje promenljive utiču jedna na drugu, a koje postaju nezavisne u prisustvu odredjene evidencije.

MAP upit (Maximum A Posteriori) Pored računanja kompletne raspodele verovatnoće, često nas zanima samo **najverovatnija vrednost** odredjene promenljive, uz zadatu evidenciju. U tom slučaju koristi se tzv. **MAP upit** (*Maximum A Posteriori*).

Listing 9: Primer MAP upita

```
# Primer MAP upita
inference = VariableElimination(model)
```

```
res = inference.map_query(['covid'], evidence={'X-ray': 'LOS
    ', 'kasalj': 'NE'})
print(res)
```

MAP upit ne vraća celu raspodelu verovatnoće, već onu vrednost promenljive koja ima **najveću uslovnu verovatnoću** u odnosu na zadatu evidenciju.

Formalno, MAP upit odredjuje:

$$\arg\max_{x} P(x \mid \text{evidence})$$

U ovom primeru, računa se koja vrednost promenljive `covid` (da li `DA` ili `NE`) ima najveću verovatnoću pod uslovom da je `X-ray` = `LOS` i `kasalj` = `NE`.

Rezultat upita je konkretna vrednost promenljive, na primer:

```
{'covid': 'DA'}
```

što znači da je, prema modelu, najverovatnije da pacijent ima COVID u datim uslovima.

Za razliku od standardnog upita `query()`, koji vraća celu raspodelu verovatnoće, `map_query()` daje samo najverovatniji ishod, što je često korisno u situacijama kada je potrebno doneti konkretnu odluku ili klasifikaciju.

Analiza nezavisnosti u modelu Jedna od važnih prednosti Bajesovih mreža jeste mogućnost automatskog odredjivanja zavisnosti i nezavisnosti izmedju promenljivih na osnovu strukture grafa. Biblioteka `pgmpy` pruža nekoliko korisnih funkcija za analizu ovih odnosa.

Listing 10: Analiza nezavisnosti i zavisnosti u modelu

```
# Pronadjene nezavisnosti u grafu
model.get_independencies()

# Pronadjene lokalne nezavisnosti vezane za promenljivu "
    kasalj"
model.local_independencies('kasalj')

# Pronadjene aktivne putanje zavisnosti u odnosu na cvor "
    kasalj"
model.active_trail_nodes('kasalj')

# Aktivne putanje kada je X-ray opazen
model.active_trail_nodes('kasalj', observed=['X-ray'])
```

Funkcija `get_independencies()` vraća skup svih nezavisnosti koje važe u modelu, izvedenih na osnovu strukture grafa. Ove nezavisnosti su posledica **d-separacije**, odnosno načina na koji su čvorovi povezani.

Funkcija `local_independencies('kasalj')` daje lokalne nezavisnosti za datu promenljivu. U našem modelu, `kasalj` je direktno povezan samo sa promenljivom `X-ray`, pa je nezavisan od ostalih promenljivih kada se uslovimo na odgovarajuće čvorove (npr. potomke ili roditelje u grafu).

Funkcija `active_trail_nodes('kasalj')` vraća sve čvorove koji su **dostupni putem aktivnih putanja** iz čvora `kasalj`. To su čvorovi na koje `kasalj` može da utiče (ili od kojih može da zavisi) kroz graf, bez zadate evidencije. U našem slučaju, to uključuje `X-ray` i dalje `covid`.

Kada uvedemo evidenciju:

```
model.active_trail_nodes('kasalj', observed=['X-ray'])
```

situacija se menja. Čvor `X-ray` je sada opažen, čime se **zatvara putanja** izmedju `kasalj` i `covid`. To znači da `kasalj` više ne utiče na `covid` kada znamo vrednost promenljive `X-ray`. Drugim rečima, ove dve promenljive postaju **uslovno nezavisne**.

Intuitivno, kada znamo rezultat rendgenskog snimka, dodatna informacija o kašlju više ne donosi novu informaciju o bolesti, jer je uticaj kašlja već "ugradjen" u promenljivu `X-ray`.

Ove funkcije omogućavaju da se struktura modela analizira bez eksplicitnog računanja verovatnoća, što je posebno korisno za razumevanje načina na koji se informacije propagiraju kroz mrežu.