Universitatea Tehnică „Gheorghe Asachi” din Iaşi

Facultatea de Automatică și Calculatoare

Sistem de Analiză a Performanței Academice & Recomandări utilizând Inferența în Rețele Bayesiene

Proiect realizat de:

Avramescu Andrei

Popa Alexandru Serban

Surdu Gabriel Cosmin

1. Descrierea problemei considerate

În acest proiect se analizează problema modelării incertitudinii în context academic, folosind o rețea bayesiană pentru a determina probabilitățile asociate performanței unui student. Problema modelează factorii care influențează nota unui student și, implicit, scrisoarea de recomandare pe care acesta o primește.

Avem o rețea cauzală definită de următoarele variabile:

- **Cauze (Noduri părinte):** *Studiază* (dacă studentul a învățat sau nu) și *Dificultate* (cât de greu este examenul).
- **Efect intermediar:** *Nota* (influențată direct de cât a studiat studentul și de dificultatea examenului).
- **Efect final:** *Recomandare* (influențată exclusiv de nota obținută).

Scopul aplicației este de a răspunde la întrebări probabilistice de tipul:

1. **Predicție:** Dacă știm că un student a studiat și examenul este ușor, care este probabilitatea să primească o recomandare bună?
2. **Diagnosticare:** Dacă știm că un student a luat o notă mică, care este probabilitatea ca examenul să fi fost greu sau ca el să nu fi învățat?

Aplicația permite încărcarea oricărei structuri de rețea bayesiană dintr-un fișier text și interogarea oricărui nod, având sau nu dovezi (evidențe) setate pentru celelalte noduri.

2. Aspecte teoretice privind algoritmul

Pentru rezolvarea problemei s-a utilizat **Inferența prin Enumerare** în rețele bayesiene. O rețea bayesiană este un graf orientat aciclic (DAG) în care nodurile reprezintă variabile aleatoare, iar arcele reprezintă dependențe probabilistice condiționate.

Concepte cheie:

1. **Probabilitatea Condiționată și Teorema lui Bayes**: Relația fundamentală utilizată este Teorema lui Bayes, care ne permite să actualizăm probabilitatea unei ipoteze pe baza unor noi evidențe. De asemenea, rețeaua se bazează pe regula lanțului (Chain Rule) pentru probabilități commune

2. **Inferența prin Enumerare (Enumeration Inference)**: Algoritmul implementat calculează distribuția de probabilitate a unei variabile de interogare, dată fiind o mulțime de variabile de evidență, sumând peste toate variabilele ascunse (cele care nu sunt nici de interogare, nici de evidență).

Algoritmul recursiv parcurge nodurile în ordine topologică. Dacă un nod este variabilă de evidență, se folosește valoarea sa fixată. Dacă este variabilă ascunsă, se iterează prin toate valorile posibile și se sumează probabilitățile ponderate.

3. Modalitatea de rezolvare

Proiectul este implementat în limbajul **C#** și este structurat modular pentru a separa logica de inferență de structura rețelei și de interfața cu utilizatorul.

Organizarea proiectului:

- **Core:** Conține entitățile de bază.
  - `Node`: Reprezintă un nod din rețea, având nume, valori posibile, părinți și Tabela de Probabilități Condiționate (CPT).
  - `BayesianNetwork`: Gestionează lista de noduri și oferă funcționalități de căutare și sortare topologică.
  - `Evidence`: Stochează valorile observate (fixate) de utilizator.
- **IO:**
  - `NetworkParser`: Este responsabil de citirea fișierului text (format personalizat) și construirea grafului în memorie.
- **Inference:**
  - `EnumerationInference`: Implementează efectiv algoritmul recursiv `EnumerateAll` și funcția de interogare `EnumerationAsk`.
- **Program.cs:** Interfața tip consolă care permite utilizatorului să încarce fișiere, să seteze evidențe și să interogheze noduri.

4. Listarea părților semnificative din codul sursă

4.1. Algoritmul de Inferență (EnumerationInference.cs)

Metoda EnumerateAll implementează recursivitatea necesară pentru a suma peste variabilele ascunse.

```
private double EnumerateAll(
    List<Node> vars,
    Dictionary<string, string> evidence)
{
    if (vars.Count == 0)
    {
        return 1.0;
    }
    Node Y = vars[0];
    var rest = vars.Skip(1).ToList();

    if (evidence.ContainsKey(Y.Name))
    {
        double prob = Y.Probability(
            evidence[Y.Name],
            evidence
        );
```

```
            return prob * EnumerateAll(rest, evidence);
        }
        else
        {
            double sum = 0.0;

            foreach (var y in Y.Values)
            {
                var extendedEvidence = new Dictionary<string, string>(evidence);
                extendedEvidence[Y.Name] = y;

                double prob = Y.Probability(y, extendedEvidence);

                sum += prob * EnumerateAll(rest, extendedEvidence);
            }

            return sum;
        }
    }
```

4.2. Calculul Probabilității unui Nod (Node.cs)

Această metodă extrage probabilitatea corectă din Tabela de Probabilități Condiționate (CPT) pe baza stării părinților.

```
    public double Probability(string value, Dictionary<string, string> evidence)
    {
        // Daca nodul nu are parinti, folosim cheia vida ""
        if (Parents.Count == 0)
        {
            if (!CPT.ContainsKey(""))
            {
                throw new Exception("CPT missing root entry for node " + Name);
            }

            return CPT[""][value];
        }

        // Construim cheia pentru CPT pe baza parintilor
        var keyParts = new List<string>();

        foreach (var parent in Parents)
        {
            if (!evidence.ContainsKey(parent.Name))
            {
```

```
            throw new Exception(
                "Missing evidence for parent " + parent.Name + " of node " + Name
            );
        }

        keyParts.Add(parent.Name + "=" + evidence[parent.Name]);
    }

    string key = string.Join(",", keyParts);

    if (!CPT.ContainsKey(key))
    {
        throw new Exception(
            "CPT entry not found for key " + key +
            " in node " + Name
        );
    }

    return CPT[key][value];

}
```

4.3. Parsarea Fișierului (NetworkParser.cs)

Secțiunea care interpretează structura CPT din fișierul text, gestionând formatele complexe cu mai mulți părinți.

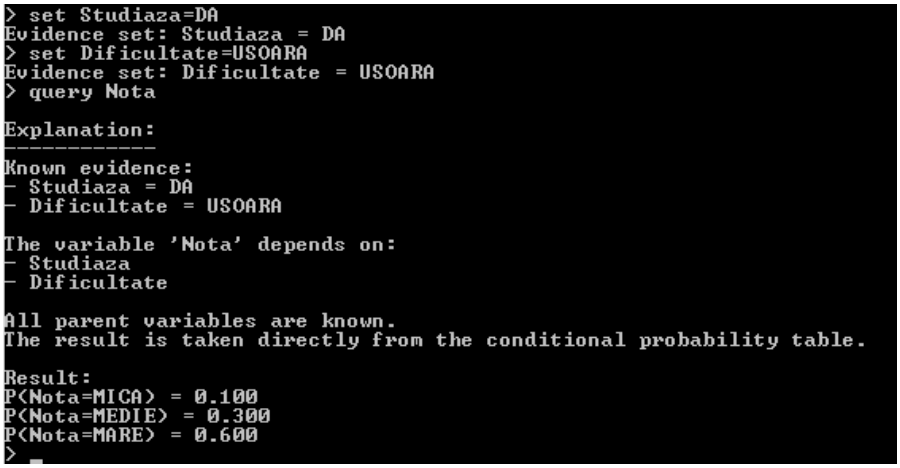
```
while (index < lines.Count)
{
    string line = lines[index];
    if (line.EndsWith(":") && !line.Contains("|"))
    {
        string nodeName = line.Replace(":", "");
        Node node = nodes[nodeName];
        node.CPT[""] = new Dictionary<string, double>();

        index++;
        while (index < lines.Count && lines[index].Contains("="))
        {
            var parts = lines[index].Split('=');
            string value = parts[0].Trim();
            double prob = double.Parse(parts[1].Trim());

            node.CPT[""][value] = prob;
            index++;
        }
```

```
    }
    else if (line.Contains("|"))
    {
        var headerParts = line.Split('|');
        string nodeName = headerParts[0].Trim();
        Node node = nodes[nodeName];

        index++;
        while (index < lines.Count && lines[index].Contains("->"))
        {
            var parts = lines[index].Split("->");
            string parentConfig = parts[0]
                .Replace(" ", "")
                .Trim();


            var probParts = parts[1].Split(',');
            var dist = new Dictionary<string, double>();

            foreach (var p in probParts)
            {
                var pv = p.Split(':');
                string value = pv[0].Trim();
                double prob = double.Parse(pv[1].Trim());
                dist[value] = prob;
            }

            node.CPT[parentConfig] = dist;
            index++;
        }
    }
    else
    {
        index++;
    }

}
```

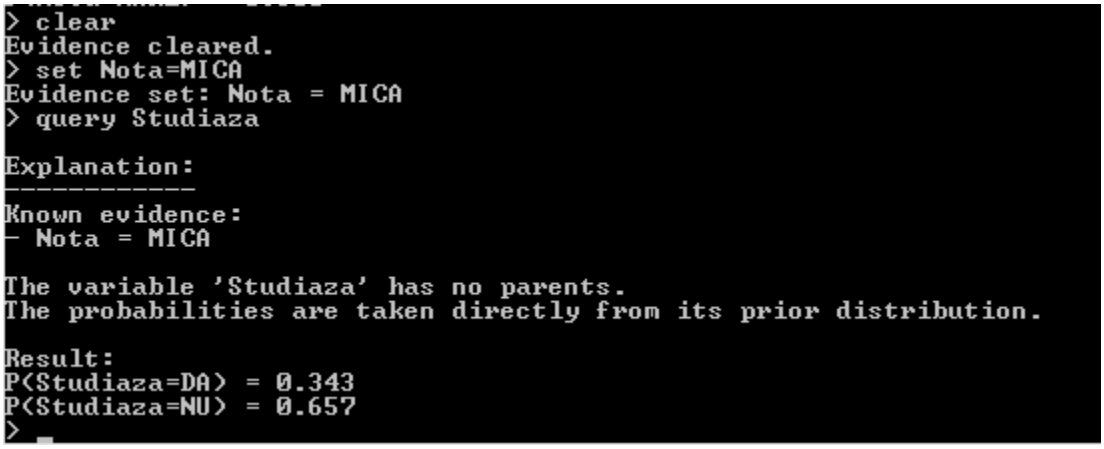
5. Rezultatele obținute

Programul a fost testat pe două rețele diferite pentru a valida generalitatea soluției.

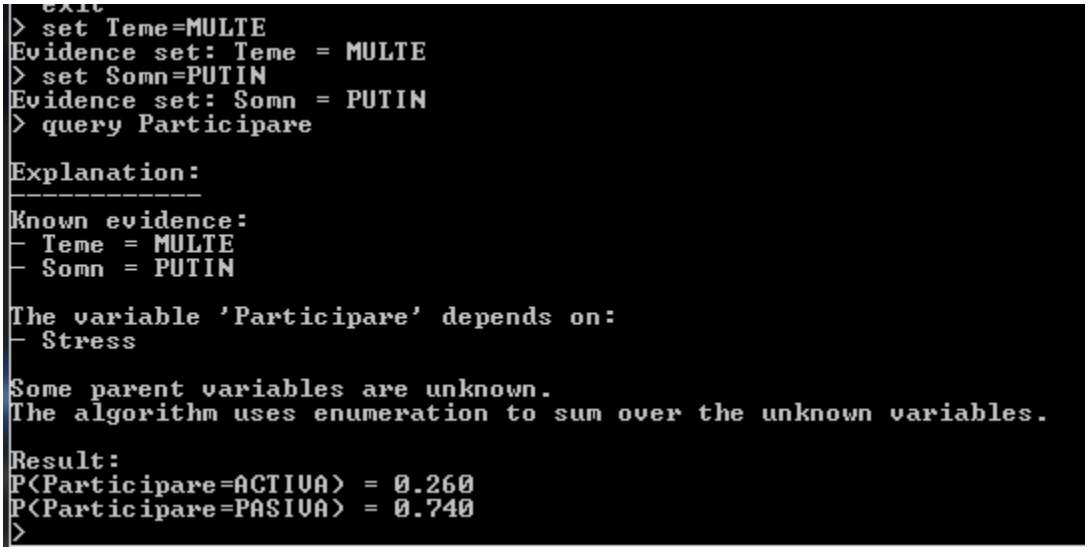
În absența oricărei dovezi, algoritmul calculează probabilitatea marginală a notei bazându-se doar pe distribuțiile a-priori ale dificultății și studiului.



Setând evidențele ca studentul a studiat (Studiaza=DA) și examenul este ușor (Dificultate=USOARA), probabilitatea pentru o notă mare crește semnificativ (la 0.600), conform așteptărilor logice definite in CPT.



Observăm fenomenul de diagnosticare. Știind că nota este mică, algoritmul calculează probabilitatea ca studentul să fi studiat. Vedem că este mult mai probabil (Studiaza=NU) ca studentul să nu fi învățat.



S-a testat o a doua rețea care modelează echilibrul studentului. În scenariul considerat, am setat evidențele pentru un volum mare de teme (Teme=MULTE) și odihnă insuficientă (Somn=PUTIN). Rețeaua a inferat corect prin propagarea probabilităților (prin nodul intermediar de Stress) că șansele ca studentul să aibă o participare activă la cursuri scad drastic (la aproximativ 20%), rezultând o participare preponderent PASIVA. Aceasta demonstrează capacitatea programului de a gestiona scenarii academice diverse.

6. Concluzii

În urma implementării acestui proiect, s-au desprins următoarele concluzii:

- Rețelele Bayesiene reprezintă o metodă robustă de a modela incertitudinea și relațiile cauzale complexe.
- Algoritmul de inferență prin enumerare, deși exponențial ca timp de execuție în cazul cel mai defavorabil (în funcție de numărul de variabile), este exact și ușor de implementat pentru rețele de dimensiuni mici și medii.
- Separarea structurii rețelei (fișier text) de logica programului permite reutilizarea codului pentru orice domeniu (medical, academic, trafic etc.), atâta timp cât datele sunt formatate corect.
- Sistemul realizat poate oferi explicații (prin afișarea părinților și a tipului de calcul) care ajută utilizatorul să înțeleagă de ce a fost obținut un anumit rezultat probabilistic.

7. Bibliografie

1. Florin Leon, Inteligență Artificială - Laborator, http://florinleon.byethost24.com/lab_ia.html
2. Russell, S., & Norvig, P., Artificial Intelligence: A Modern Approach.
3. Cursuri de Inteligență Artificială, Facultatea de Automatică și Calculatoare, Iași.

8. Contribuția fiecărui membru al echipei

Avramescu Andrei:

- A proiectat arhitectura modulară a aplicației și structura claselor din modulul Core (Node, BayesianNetwork, Evidence), stabilind relațiile dintre entități.

- A definit standardul pentru fișierele de intrare și structura tabelelor de probabilități (CPT), asigurând consistența datelor.

Surdu Gabriel Cosmin:

- A conceput și implementat scenarii de testare suplimentare, inclusiv rețeaua stress.txt cu o topologie complexă (tip "X" cu ramificații multiple) pentru a demonstra versatilitatea soluției.
- A realizat validarea matematică a algoritmului, comparând rezultatele oferite de program cu cele calculate manual pentru a asigura precizia.

Popa Alexandru Serban:

- A dezvoltat modulul IO (NetworkParser), implementând mecanisme robuste de citire și tratare a erorilor pentru fișierele de configurare.
- A realizat documentația tehnică a proiectului, explicând fundamentul teoretic și detaliind structura problemei.