

Inferencia en redes bayesianas

Contenido

1-	Descripción del ejemplo elegido	2
2-	Descripción de la representación escogida para manejar la red y sus tablas.....	3
3-	Descripción informal del diseño del algoritmo de eliminación de variables, las heurísticas y el recomendador	3
3.1	Algoritmo de eliminación de variables (VE)	3
3.2	Heurísticas	4
3.3	Recomendador	4
4-	Sección de experimentación con ejemplos concretos del uso del algoritmo y del recomendador.....	8
5-	Listado de las referencias que se hayan consultado	9

1- Descripción del ejemplo elegido

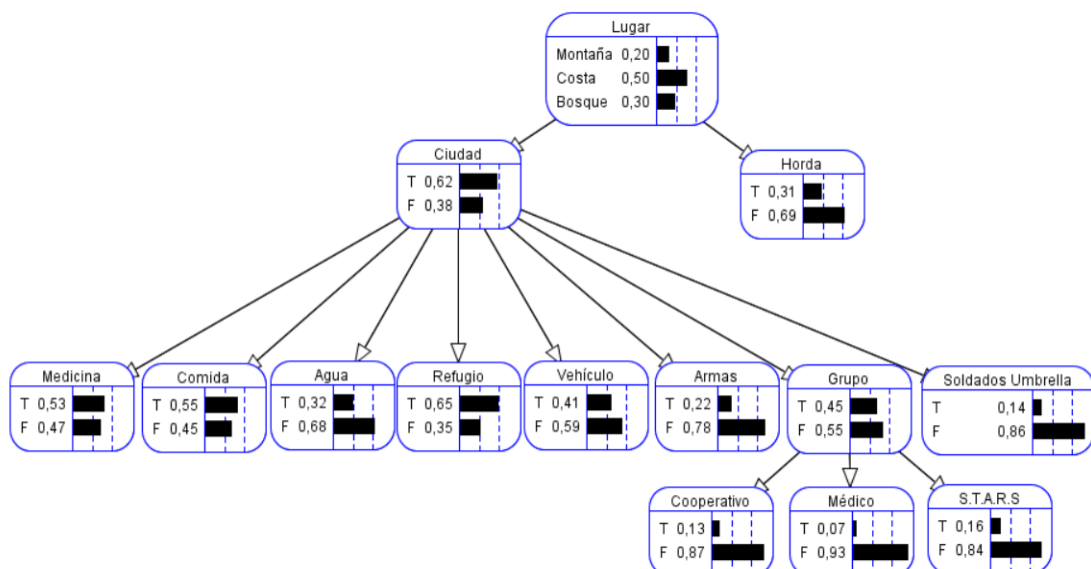
Para afrontar la realización de este trabajo hemos decidido utilizar un supuesto basado en el *lore* de Resident Evil. El bioterrorismo, una amenaza existente en el mundo real, es el pilar sobre el que se desarrolla este mundo. Un virus mutágeno es liberado por accidente (o no) provocando una epidemia global y como producto de ella, el apocalipsis.

Sin entrar en detalles sobre este universo, la consecuencia es que, como en cualquier supuesto post apocalíptico, hay supervivientes que se enfrentan a continuas amenazas entre las que se encuentran, en este caso, los propios mutantes los cuales pierden la condición de humano convirtiéndose en depredadores, la falta de suministros, peligros diversos del entorno y, como no, el propio ser humano.

El objetivo de nuestra red es, basándonos en unas probabilidades lo más cercanas a la realidad posibles, surtir de información sobre la posibilidad de sobrevivir dadas ciertas circunstancias.

A pesar de que dicho mundo es ficticio, nuestra justificación a las probabilidades otorgadas es el sentido común y el pragmatismo ya que el único elemento irreal es la existencia del propio virus. El resto de amenazas (incluyendo la existencia de depredadores con unas características muy concretas) son emulables con cierto grado de realismo. Por ello podemos afirmar que nuestra red es, dentro de la ficción, fidedigna.

A continuación, la representación de dicha red con sus probabilidades.



2- Descripción de la representación escogida para manejar la red y sus tablas

Las herramientas elegidas para la representación de la red y sus tablas son las siguientes.

Para la representación (el gráfico del punto uno, descripción) hemos utilizado *Believe and Decission Network Tool* (versión 5.1.10). Dicha herramienta es un ejecutable *Java* sencillo de usar que permite la construcción de redes bayesianas y la realización de consultas sobre la misma obteniendo las correspondientes tablas asociadas y permitiendo configurar la consulta de modo que vemos el desarrollo paso a paso de la misma y el funcionamiento del algoritmo.

Por otro lado, para su implementación y utilización mediante nuestro propio código en python hemos utilizado las herramientas empleadas en las clases prácticas. El editor *Jupyter Notebook*, que es un entorno de desarrollo secuencial lanzado mediante el navegador de internet. Permite la edición de archivos tanto locales como alojados en un servidor e importando el paquete *PGMPY*, que permite trabajar con redes bayesianas.

Para facilitar la elaboración del código hemos añadido una extensión de autocompletado a nuestro interprete Jupyter que muestra las funciones y acciones disponibles para los tipos de datos declarados (variables, constantes etcétera).

3- Descripción informal del diseño del algoritmo de eliminación de variables, las heurísticas y el recomendador

3.1 Algoritmo de eliminación de variables (VE)

Para diseñar el algoritmo de eliminación de variables se han seguido los siguientes pasos:

1-Función “descarta”: se encarga de descartar las variables irrelevantes. Recibe como parámetros un modelo de red, una lista de consultas y una lista de evidencias. Y devuelve una lista con las variables irrelevantes.

2- Función “factores_iniciales”: se encarga de construir los factores iniciales. Recibe como parámetros un modelo, una lista de variables descartadas (utilizando para ello la función anterior), y un diccionario de evidencias (cuya clave es el nombre de la variable y el valor es el valor de esa variable en la evidencia). Y devuelve los factores generados.

3- Función “calcula_elimina”: se encarga de calcular las variables ocultas que hay que eliminar. Recibe un conjunto de factores, una lista de consultas y una lista de evidencias. Y devuelve una lista de las variables a eliminar. Esta función se ha implementado a parte para evitar la redundancia de código, ya que era necesario usarla en varias funciones.

4- Función “eliminar_variables”: se encarga de eliminar las variables del conjunto de factores. Recibe como parámetros un conjunto de factores, una lista de consultas y una lista de evidencias. Hace uso de la función anterior “calcula_elimina” para obtener la lista de las variables que necesita eliminar. Devuelve un conjunto de factores con las variables ya eliminadas.

5- Función “mn_factores”: se encarga de multiplicar los factores restantes y normalizar el resultado. Recibe un conjunto de factores. Y devuelve un factor ya normalizado.

Por último, se ha usado la herramienta “**VariableElimination**” (que ya dispone de la implementación del proceso de eliminación de variables) con el objeto de comprobar el correcto resultado obtenido con las funciones implementadas anteriormente.

3.2 Heurísticas

-Min Degree: dado un conjunto de factores, la variable que se decide eliminar es la que está conectada con menos variables en el conjunto de factores. Decimos que una variable está conectada con otra respecto de un conjunto de factores, cuando ambas aparecen en un mismo factor del conjunto.

-Min Fill: dado un conjunto de factores, la variable que se decide eliminar es aquella que al eliminarse introduciría menos conexiones nuevas (conexiones en el sentido explicado en el punto anterior).

-Min Factor: dado un conjunto de factores, la variable que se decide eliminar es aquella que al eliminarse produciría el factor más pequeño (es decir, el que menos entradas tendría).

3.3 Recomendador

Nuestro recomendador utiliza el algoritmo de inferencia aproximada **muestreo por rechazo**. Es una función que recibe como parámetros la red, la consulta y un conjunto de candidatas a ser evidencias. El resultado de la ejecución es la recomendación de una nueva evidencia. Dicha evidencia será la que mayores cambios aporte a la consulta pudiendo cambiar totalmente los resultados y ayudando a decidir al usuario acerca de las decisiones que debe tomar en base a las probabilidades aportadas por la red.

A continuación, se incluye el código de dicho recomendador junto con las funciones que necesita para llevar a cabo su trabajo:

```
import random

#Función que devuelve una probabilidad concreta del modelo
def seleccionar_probabilidad(cpd, valor, evidencia):
    # Buscamos los padres de una evidencia
    padres = cpd.evidence if cpd.evidence else []
    # Buscamos los valores asociados a los padres
    valores_evidencia = tuple(evidencia[var] for var in padres)
    # Buscamos los valores de la distribución de probabilidad conjunta
    return cpd.values[valor][valores_evidencia]

#Función que genera un valor aleatorio según la cardinalidad de la variable
def generar_valor_aleatorio(cardinalidad, probabilidades):
    p = random.random()
    acumuladas = 0
    for valor in range(cardinalidad):
        acumuladas += probabilidades[valor]
        if p <= acumuladas:
            return valor
```

```
#Función que genera una muestra aleatoria dada una red bayesiana
#Esta función usa las dos anteriores (seleccionar_probabilidad y generar_valor_aleatorio)
def muestra_aleatoria(red):
    variables = networkx.topological_sort(red)
    muestra = {}
    for variable in variables:
        tabla = Modelo_residentevil.get_cpds(variable)
        #TODO: con Lugar no funciona bien, hay que arreglarlo
        if variable != 'Lugar':
            cardinalidad = tabla.variable_card
            probabilidades = [seleccionar_probabilidad(tabla, i, muestra)
                               for i in range(cardinalidad)]
            valor = generar_valor_aleatorio(cardinalidad, probabilidades)
        else:
            valor = 0
        muestra[variable] = valor

    return muestra
```

```
import itertools

#Función que devuelve las frecuencias aplicando el algoritmo de muestreo por rechazo
def muestreo_con_rechazo(red, consulta, evidencia, N):
```

```

    muestras = [muestra_aleatoria(red) for i in range(N)] # Me generará una muestra de N valores

    # Ahora nos quedamos con aquellas en las que la muestra y la evidencia coinciden
    muestrasValidas = []
    for muestra in muestras:
        if all(muestra[variable] == evidencia[variable]
               for variable in evidencia):
            muestrasValidas.append(muestra)

    cardinalidades = [red.get_cpds(variable).variable_card
                       for variable in consulta]

    # Calculamos las combinaciones
    combinaciones = itertools.product(*(range(i)
                                           for i in cardinalidades))

    # Ahora contamos las combinaciones, es decir, el n° de veces que ocurre cada cosa
    # Recuerda: es un diccionario
    frecuencias = {combinacion : 0
                   for combinacion in combinaciones}

    for muestra in muestrasValidas:
        combinacion = tuple(muestra[variable]
                             for variable in consulta)
        frecuencias[combinacion] += 1

    #print (combinaciones)
    #print (frecuencias)
    #print (len(muestras))
    #print (len(muestrasValidas))
    return frecuencias

```

```

def recomendador(modelo, consulta, evidencias):
    k = 0
    suma = 0
    i = 0
    candidatas = []
    e = list(evidencias.keys())
    nevidencias = {}
    res={}

    descartes = descarta(modelo, consulta, e)
    nodos = modelo.nodos()

    #Calculamos las candidatas
    for n in nodos:
        if (n not in consulta) and (n not in e) and (n not in descartes):
            candidatas.append(n)

    #Aplicamos muestreo por rechazo con las candidatas con valor 0
    for c in candidatas:
        nevidencias.update(evidencias)

```

```

        #Añadimos al diccionario de evidencias la candidata con val
or 0
        nevidencias[c] = 0
        #Llamamos al algoritmo muestreo con rechazo
        frecuencias = muestreo_con_rechazo(modelo, consulta, nevide
ncias, 10000)
        #Normalizamos
        suma = frecuencias[(0,)] + frecuencias[(1,)]
        pno = frecuencias[(0,)] / suma
        psi = frecuencias[(1,)] / suma
        #Calculamos la diferencia
        dif = abs(pno-psi)
        #Asignamos la nueva candidata si la diferencia es mayor
        if(k < dif):
            k = dif
            i = 0
            r = c
        nevidencias.clear()

        #TODO: esto hay que arreglarlo, ya que Lugar no lo coge bien, t
engo que borrarla
        candidatas.remove('Lugar')

        #Aplicamos muestreo por rechazo con las candidatas con valor 1
        for c in candidatas:
            nevidencias.update(evidencias)
            #Añadimos al diccionario de evidencias la candidata con val
or 1
            nevidencias[c] = 1
            #Llamamos al algoritmo muestreo con rechazo
            frecuencias = muestreo_con_rechazo(modelo, consulta, nevide
ncias, 10000)
            #Normalizamos
            suma = frecuencias[(0,)] + frecuencias[(1,)]
            pno = frecuencias[(0,)] / suma
            psi = frecuencias[(1,)] / suma
            #Calculamos la diferencia
            dif = abs(pno-psi)
            #Asignamos la nueva candidata si la diferencia es mayor
            if(k < dif):
                k = dif
                i = 1
                r = c
            nevidencias.clear()

        #Devolvemos la variable candidata cuya diferencia es mayor como
recomendación

        res[r] = i
        return res

```

4- Sección de experimentación con ejemplos concretos del uso del algoritmo y del recomendador

La consulta utilizada como ejemplo durante el desarrollo del algoritmo ha sido: la probabilidad de que **Refugio** sea **verdadero**, sabiendo que **Ciudad** y **Cooperativo** son **verdaderos**.

```
#Probando la función muestra aleatoria  
muestra_aleatoria(Modelo_residentevil)
```

```
{'Agua': 0,  
'Armas': 0,  
'Ciudad': 0,  
'Comida': 0,  
'Cooperativo': 0,  
'Grupo': 0,  
'Horda': 0,  
'Lugar': 0,  
'Medicina': 1,  
'Refugio': 0,  
'Soldados umbrella': 1,  
'Vehiculo': 0}
```

```
#Probando la función muestreo por rechazo, que devuelve las frecuencias  
random.seed(12345)  
print('Las frecuencias son:',muestreo_con_rechazo(Modelo_residentevil, ['Refugio'], {'Ciudad': 1, 'Cooperativo': 1}, 10000))
```

Las frecuencias son: {(0,): 818, (1,): 2340}

A fin de probar el recomendador se realizan las siguientes consultas dando los correspondientes resultados:

```
#Probando la función recomendador  
vr = recomendador(Modelo_residentevil, ['Refugio'], {'Ciudad': 1, 'Cooperativo': 1})  
print('La variable recomendada es:',vr)
```

La variable recomendada es: {'Grupo': 0}

5- Listado de las referencias que se hayan consultado

<http://pgmpy.org/>

<https://stackoverflow.com/>

<http://aispace.org/bayes/>