MODELOS GRÁFICOS PROBABILISTICOS

Mario De Los Santos Hernández

Proyecto 2. Naive Bayes

Implement the Forward algorithm for estimating the probability of a sequence of observations given the model. The program should work for any discrete HMM and any observation sequence.

- Input: number of states, number of observations, prior probability vector, transition matrix, observation matrix.
- Input: an observation sequence (user gives it from the keyboard).
- Output: probability of the sequence.

Test on the coins example (problem 12 textbook).

Extra points: investigate other HMMs and test on these other cases.

1. Reporte

Como parte del proceso de asimilación de los distintos modelos gráficos probabilistas, en esta ocasión trabajamos con una implementación de los *Hidden Márkov Models*, los cuales son una cadena de Márkov donde los estados nos on directamente observables. El siguiente proyecto representa la aplicación del algoritmos iterativo *Forwarding* mostrado por el autor de libro de referencia y profesor del curso. La implementación ha sido probada con dos ejemplos; el primero demostrado en el libro y tomado como referencia en la tarea 6, la cual ha servido de guía en esta asignación.

La versión actual del proyecto se puede encontrar en el siguiente repositorio, esto con la finalidad de facilitar tu acceso en futuras ocasiones, y ha sido desarrollado con un paradigma orientado a objetos usando el lenguaje C++.

- GitHub: Hidden Markov Models – Forwarding Algorithm

- Ejemplo 1

$$\Pi = \begin{array}{cc} M_1 & M_2 \\ \hline 0.5 & 0.5 \end{array} \qquad A = \begin{array}{c|cc} & M_1 & M_2 \\ \hline M_1 & 0.5 & 0.5 \\ M_2 & 0.5 & 0.5 \end{array} \qquad B = \begin{array}{c|cc} & M_1 & M_2 \\ \hline H & 0.8 & 0.2 \\ T & 0.2 & 0.8 \end{array}$$

Figura 1. Parámetros del ejercicio 1.

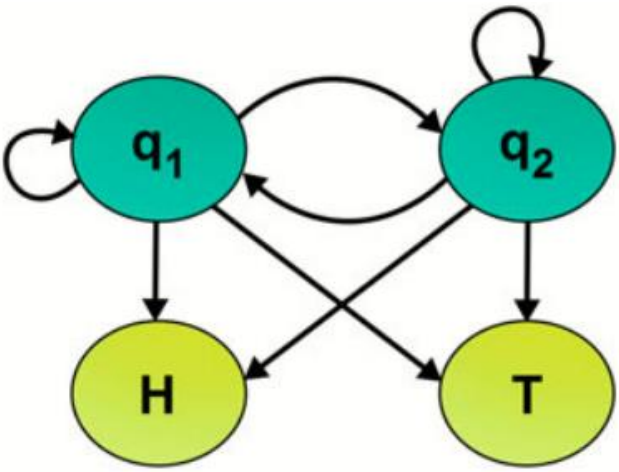

Figura 2. Diagrama del ejemplo 1. Dos estados con dos observaciones.

Una vez introducidos dichos datos se han probado distintas secuencias, obteniendo los resultados esperados, estás secuencias se introducen por el momento de manera manual al código por medio de la consola, con ello, las siguientes figuras demuestran el concepto.

```
Insert Probability Vector:
S0
0.5
S1
0.5
Insert Observation Matrix:
0.8
0.2
0.2
0.8
Insert Transition Matrix:
0.5
0.5
0.5
0.5
Define sequence in numerical values
0
0
1
1
Resultado: 0.0625
```

Figura 3. Secuencia HHTT, donde H=0 y T=1.

Otras secuencias probadas:

- TTHH = 0.0625
- THT = 0.125

- Ejemplo 2

Con la finalidad de comprobar el funcionamiento del sistema, hemos probado la implementación con un ejercicio propuesto por un blog de nombre *towards data science*, el cual brinda múltiples ejercicios para practicar la aplicación de la teoría detrás de los HMMs, con ello se han utilizado los siguientes datos:

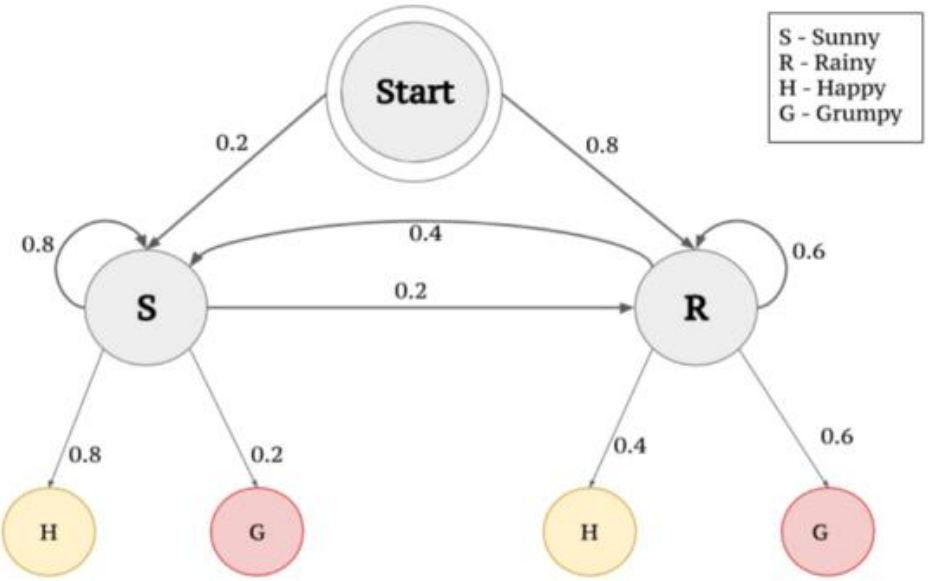

Figura 4. Ejemplo 2, Diagrama completo del problema.

Partiendo del la figura 4, se han extraído las siguientes tablas para poder aplicar al algoritmo.

	S0	S1
$\pi =$	0.2	0.8

		S	R
A =	S	0.8	0.4
	R	0.2	0.8

		S	R
B=	H	0.8	0.4
	G	0.2	0.6

Figura 5. Tablas de Prior probability, transición y observación.

Dados dichos datos, hemos realizado pruebas en distintas secuencias, y con ello cumplido el objetivo de buscar un ejemplo extra.

- HHG = 0.093184
- GHGG = 0.0662323
- HHGH = 0.0474317

```
Insert Probability Vector:
S0
0.2
S1
0.8
Confirmation
S0 0.2 S1 0.8 Insert Transition Matrix:
S0 to S0
0.8
S0 to S1
0.4
S1 to S0
0.2
S1 to S1
0.8
0.8 0.4
0.2 0.8
Insert Observation Matrix:
S0 to O0
0.8
S0 to O1
0.4
S1 to O0
0.2
S1 to O1
0.6
0.8 0.4
0.2 0.6
Define sequence in numerical values
0
0
1
0
Resultado: 0.0474317
```

Figura 6. Resultado de la secuencia HHGH.

- Código

La siguiente sección tiene la finalidad de mostrar las bases del código y sus partes, solo mostraran aquellas que determinan el uso de la clase, el resto de los detalles, como la parte operacional puede encontrarse detallada línea por línea en el repositorio del proyecto.

```
//Parameterized constructor
HMMs_Forward(int N_Sts, int N_Obs)
//The user should pass the number of states and observations that the problem has
{
    if(N_Sts|| N_Obs != 0) { //Verification for a valid information
        No_Observations = N_Obs; //Data hiding, information will be part of the object class
        No_states = N_Sts; //Will keep private

        //Functions to extract the matrixes with the needed information
        Probability_vector(No_states);
        Transition_matrix(No_Observations);
        Obs_matrix(No_states, No_Observations);
    }
    else
        cout<<"Invalid register"<<endl;

}
```

Figura 7. Constructor.

El constructor se encarga de definir las necesidades de la clase para operar el algoritmo, una vez creado el objeto es necesario definir el número de estados y observaciones, con ello se definirá la recolección de datos.

```
//Object creation, passing the predefined parameters,
// then you would need to fit the vectors needed to operate
HMMs_Forward T1( N_Sts: 2, N_Obs: 2);
```

Figura 8. Creación del Objeto.

El siguiente punto importante por comentar es la función de *forwarding*, la cual fue la meta del proyecto. Para ello, hemos usado de referencia el pseudocódigo proporcionado por el material del libro, el cual básicamente nos describe el comportamiento del algoritmos para su aplicación.

Algorithm 5.1 The Forward Algorithm

Require: HMM, λ; Observation sequence, O; Number of states, N; Number of observations, T

for $i = 1$ **to** N **do**
  $\alpha_1(i) = P(O_1, S_1 = q_i) = \pi_i b_i(O_1)$ (Initialization)
end for
for $t = 2$ **to** T **do**
  for $j = 1$ **to** N **do**
    $\alpha_t(j) = [\sum_i \alpha_{t-1}(i) a_{ij}] b_j(O_t)$ (Induction)
  end for
end for
$P(O) = \sum_i \alpha_T(i)$ (Termination)
return $P(O)$

Figura 9. Algoritmo HMMs Forward.

```
double Fordward_Operation(int sequence_size)
{
    double aux=0;
    double sum=0;

    //Backip matrix just to keep alpha clean of the operations until the last step in the inductive step
    double alpha_backup[100]={0};

    //Sequence definition
    cout<<"Define sequence in numerical values"<<endl;
    for(int i=0;i<sequence_size;i++)
        cin>>sequence[i];
    //Inicialization
    for(int i=0;i<No_states;i++){
        alpha[i]=Prob_vector[i]*Observation_matix[i][sequence[0]];
    }
    //Induction
    //Moving around the sequence size given by the user
    for(int t=1;t<sequence_size;t++)
    {
        //Moving thru the states
        for(int j=0;j<No_states;j++) {
            aux=0; //Reset the sum each cicle from the += operations
            //Moving thru the states
            for (int k = 0; k<No_states; k++) {
                //Summation of both last states multiplicated for the transition cost
                aux += alpha[k] * Trans_matrix[k][j];
            }
            //save the value in the array multiplication of the aux * the observation probability of the related state
            alpha_backup[j] = aux * Observation_matix[j][sequence[t]];
            /.../
        }
        //The array alpha shall be changing wich eeach iteration, taking the calculeted values,
        //Basically this is the main goal of the algorithm.
        for(int i=0;i<No_states;i++)
        {
            alpha[i] = alpha_backup[i];
        }
    }
    //Last step is to sum the branches of the array, then return the value
    for(int i=0;i<No_states;i++) {
        sum += alpha[i];
    }
    return sum;
}
```

Figura 10. Implementación del Algoritmo Forward.

- Referencias

  - Sucar, L. E. (2020). Probabilistic graphical models. Advances in Computer Vision and Pattern Recognition.London: Springer London. doi, 10(978), 2.
  -