Optimización de Redes Bayesianas por medio de un Algoritmo Genético

Carlos Manuel Rodríguez Martínez

3 de enero de 2020

1. Introducción

En este reporte se describe la implementación en el lenguaje Wolfram de un sistema de redes bayesianas optimizadas por medio de un algoritmo genético. En la sección 2 se hace una revisión la importancia de las redes bayesianas para la predicción. La sección 3 provee descripción general del algoritmo de redes bayesianas y optimización genética, cuyos detalles e implementación son explorados a profundidad en las secciones 4, 5, 6, 7 y 8. Por último se muestran los resultados y la interfaz de usuario en la sección 9.

2. Estado del arte

Las redes bayesianas son ampliamente utilizadas en contextos donde el modelado probabilístico es conveniente, esto las ha hecho muy populares no solo en el área de machine learning sino también en la toma de decisiones de negocios. Actualmente existen múltiples suites tanto abiertas como comerciales que implementan este tipo de algoritmos. Un ejemplo es Weka, un entornos de análisis de datos que dentro de sus algoritmos posee varias implementaciones de redes bayesianas que son optimizables con varios algoritmos de búsqueda entre estos HillClimber, K2, GeneticSearch, etc. Asimismo existe software más especializado en algoritmos bayesianos como Tetrad que permite hacer inferencia bayesiana sobre la red obtenida del proceso de ajuste.

3. Metodología

3.1. Redes bayesianas

Una red bayesiana es un modelo gráfico que codifica las dependencias probabilísticas entre múltiples variables. Consiste de un conjunto distribuciones de probabilidad condicional y un grafo acíclico dirigido (DAG por sus siglas en inglés) en el cual los nodos representan las variables aleatorias y las aristas representan las dependencias entre las variables [2].

Un DAG es un conjunto de nodos y aristas los cuales, tal como sugiere su nombre, no forman ningún ciclo cerrado. Considere el siguiente grafo acíclico dirigido de cinco nodos que se muestra en la figura 1.

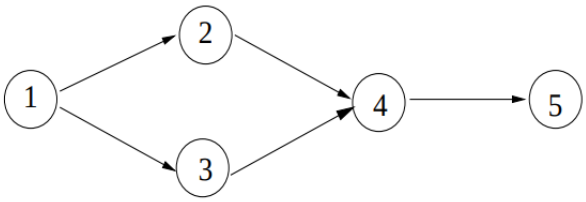


Figura 1: Un grafo acíclico dirigido.

En este grafo se dice que el nodo 1 es un padre de los nodos 2 y 3, y que los nodos 2 y 3 son hijos del nodo 1. La figura 1 representa la probabilidad conjunta $p(x_1, x_2, \ldots, x_5)$ para las variables aleatorias $X_1, X_2, \ldots X_5$ que se puede escribir como

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2, x_3)p(x_5|x_4).$$

En general esta relación se puede resumir como

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p\left(x_i|\Pi_i\right),$$

donde Π_i denota el conjunto de variables aleatorias correspondientes a nodos padres del nodo i.

3.2. Algoritmos genéticos

Los Algoritmos Genéticos (GA por sus siglas en inglés) son métodos adaptativos que pueden ser usados para resolver problemas de búsqueda y optimización y están fuertemente inspirados en el proceso biológico de evolución. Por este motivo mucha de la terminología asociada se toma de las ciencias biológicas, sin embargo las entidades descritas por la terminología son mucho más simples que su contraparte natural. Los componentes básicos que poseen casi todos los algoritmos genéticos son:

- una función para evaluar la aptitud
- una población de genomas
- selección de los genomas aptos para reproducción
- operación de cruza para producir la siguiente generación de genomas
- mutación aleatoria de los genomas

En el contexto de redes bayesianas, el genoma representa la estructura del grafo dada por su matriz de adyacencia. La optimización genética se realiza evaluando los genomas por medio de una función que determine la puntuación de la red bayesiana. El motivo por el cual es necesario utilizar una búsqueda heurística en vez de una búsqueda exhaustiva tiene que ver con la cantidad

posible de configuraciones de DAGs dado por la fórmula de Robinson (1977) [5] que devuelve la cantidad de grafos DAG dado un número n de vértices,

$$f(n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} f(n-i).$$

Para grafos con un número n de nodos la cantidad de configuraciones posibles es:

n	f(n)
1	1
2	3
3	25
4	543
5	29281
6	3 781 503
7	1 138 779 265
8	783 702 329 343
9	1 213 442 454 842 881

Lo cual es imposible evaluar exhaustivamente en un tiempo razonable para una cantidad de nodos mayor a 6.

4. Inferencia bayesiana

En una red bayesiana la inferencia se puede realizar evaluando múltiples posibles instancias y escogiendo aquella que maximice la probabilidad. Formalmente esto se define como

$$\text{Argmax}_{A,B} P(a, b|C = c).$$

Un ejemplo de inferencia se puede hacer sobre la red bayesiana de la figura 2 que representa un modelo para la base de datos Iris.

Supongamos que se observa la instancia Sepal length = 0, Sepal width = 2, Petal length = 0, Class = Iris-setosa. Para calcular la probabilidad de haber observado dicha instancia bajo este modelo primero se calculan las probabilidades marginales:

```
PrintBayesProbabilities[{1 ↦ 2, 1 ↦ 3, 2 ↦ 3, 3 ↦ 4}, irisDataset, {0, 2, 0, "Iris-setosa"}, irisAttributes]
P( x_Sepal length  = 0            )= 0.393333
P( x_Sepal width   = 2            |  x_Sepal length              = 0      )= 0.661017
P( x_Petal length  = 0            |  x_{Sepal length,Sepal width} = {0, 2} )= 1.
P( x_Class         = Iris-setosa  |  x_Petal length              = 0      )= 1.
```

Y posteriormente se calcula la probabilidad conjunta haciendo la multiplicación:

$$P\left(x_{\text{Sepal length}} = 0, x_{\text{Sepal width}} = 2, x_{\text{Petal length}} = 0, x_{\text{Class}} = \text{Iris-setosa}\right) =$$

$$P(x_{\text{Class}} = \text{Iris-setosa}|x_{\text{Petal length}} = 0) \times P(x_{\text{Petal length}} = 0|x_{\text{Sepal length}} = 0, x_{\text{Sepal width}} = 2)$$

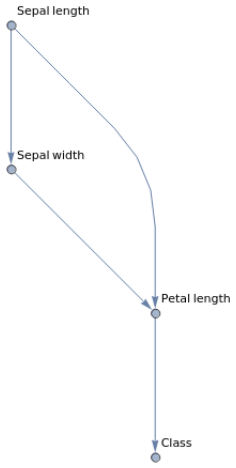


Figura 2: Modelo para el dataset Iris

$\times P(x_{\text{Sepal width}} = 2 | x_{\text{Sepal length}} = 0) \times P(x_{\text{Sepal length}} = 0) = 1{,}0 \times 1{,}0 \times 0{,}66 \times 0{,}39 = 0{,}26$

Para realizar una inferencia se evalúan todas las posibles sustituciones de la variable a predecir y se escoge aquella que maximice la probabilidad. Por ejemplo, para la misma red y la instancia Sepal length = 2, Sepal width = 0, Petal length = 1, Class =?, las posibles predicciones y probabilidades son:

```
BayesProbabilities[{1 ↦ 2, 1 ↦ 3, 2 ↦ 3, 3 ↦ 4}, irisDataset, {2, 0, 1, "?"}]
{{Iris-setosa, 0.}, {Iris-virginica, 0.000592593}, {Iris-versicolor, 0.0260741}}
```

En este caso la mejor predicción de la red es que la instancia corresponde a una Iris-versicolor.

5. Algoritmo generador de DAG

El objetivo a resolver en este problema es generar un algoritmo que enliste todos los posibles grafos acíclicos dirigidos (o DAG por sus siglas en inglés). Un DAG es un grafo dirigido que no posee ciclos, esto es, dado un vértice v no existe ninguna secuencia de transiciones que nos regrese al mismo vértice v.

5.1. Matriz de adyacencia

Un grafo está definido formalmente por una serie de vértices relacionados por nodos que especifican la relación entre los vértices. Sea V el conjunto de vértices de un grafo, se puede construir una representación matricial del grafo a partir de una matriz cuadrada A de tamaño $|V| \times |V|$ tal que sus elementos A_{ij} sean 1 cuando exista un nodo entre el V_i y V_j, y 0 cuando no exista.

Ejemplo

En la figura 4 se muestra un ejemplo de grafo DAG con su respectiva representación en forma de matriz de adyacencia. Nótese que toda matriz de

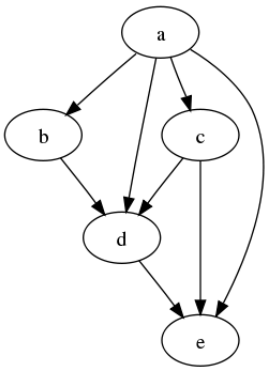
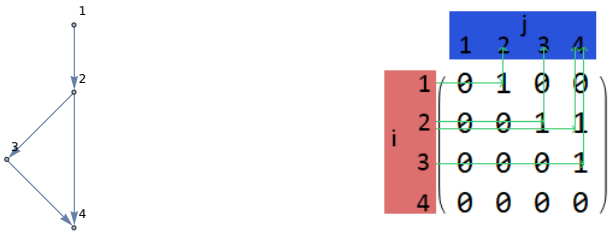


Figura 3: Ejemplo de DAG. Nótese que no existe ninguna secuencia de transiciones que comenzando desde un estado (por ejemplo el b), nos regrese al mismo.

adyacencia estrictamente triangular superior

$$A = \begin{cases} A_{ij} & 1 < j < i \\ 0 & \text{en caso contrario} \end{cases}$$

representa un DAG.



(a) Grafo DAG, definido por los vértices entre los nodos.

(b) Representación matricial del mismo grafo.

Figura 4: Ejemplo de DAG y sus representaciones.

5.2. Representaciones equivalentes de la matriz de adyacencia de un DAG

Debido a la forma en la que está construida la matriz de adyacencia es posible renombrar los índices de los vértices sin alterar la estructura del grafo. En la figura 5 se muestra el mismo grafo que en la figura 4 con la única diferencia de que se ha invertido la posición de los índices 1 y 2 en j. Esto no afecta a la estructura del grafo ya que la posición de los índices es sólo una convención. Se dice que dos digrafos acíclicos son isomorfos si existe un mapeo uno a uno entre ambos que preserva las relaciones de adyacencia.

La posibilidad de invertir índices sin afectar la estructura del grafo nos permite definir un conjunto de transformaciones que sobre la matriz de adyacencia que representan el mismo grafo.

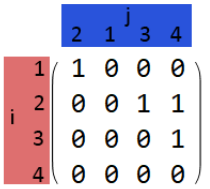


Figura 5: El mismo DAG de la figura 4 con los índices 1 y 2 en j invertidos.

Definición 1. *Sea A una matriz que representa a un grafo, se define la función de inversión de índices α, β vertical como*

$$T_{\alpha\beta V}(A) = \begin{cases} A_{i\beta} & j = \alpha \\ A_{i\alpha} & j = \beta \\ A_{ij} & j \neq \alpha, \beta \end{cases}$$

y la función de inversión de índices α, β horizontal como

$$T_{\alpha\beta H}(A) = \begin{cases} A_{\beta j} & i = \alpha \\ A_{\alpha i} & i = \beta \\ A_{ij} & i \neq \alpha, \beta \end{cases}$$

Nótese que todas las transformaciones T cumplen la propiedad de linealidad

$$T(A) + T(B) = T(A + B),$$
$$T(cA) = cT(A).$$

Ejemplo

Si

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix},$$

entonces

$$T_{12H}(A) = \begin{pmatrix} b & a & c \\ e & d & f \\ h & g & i \end{pmatrix}, \quad T_{12V}(A) = \begin{pmatrix} d & e & f \\ a & b & c \\ g & h & i \end{pmatrix},$$

5.3. Eigenvalores de la matriz de adyacencia

Supongamos que A representa la matriz de adyacencia de un grafo G, esto es, $A = A(G)$. A sólo debe tener ceros en su diagonal, de lo contrario habría ciclos de los vértices consigo mismos. Se define $B = I + A$, la cual tiene también la propiedad de ser una matriz booleana. Si el grafo G es un DAG, entonces siempre debe ser posible obtener una matriz estrictamente triangular superior por medio de una composición de transformaciones $T_{\alpha\beta V}$ y $T_{\alpha\beta H}$. Entonces los eigenvalores λ de B están dados por

$$\begin{vmatrix} 1-\lambda & x & \cdots & y \\ 0 & 1-\lambda & \cdots & z \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1-\lambda \end{vmatrix} = 0$$

cuya única solución es $\lambda = 1$.

Teorema 1. *Sea A una matriz de adyacencia que representa un grafo G, se define la transformación*

$$T_{\alpha\beta}(A) = T_{\alpha\beta H}(T_{\alpha\beta V}(A)) = T_{\alpha\beta V}(T_{\alpha\beta H}(A)),$$

y la función $\det(A)$ *que representa la determinante de A. Entonces* $\det(A) = \det(T_{\alpha\beta}(A))$.

Demostración. Cada operación de intercambio de filas o columnas tiene el efecto de invertir el signo de la determinante, esto es

$$\begin{aligned}\det\left(T_{\alpha\beta H}(A)\right) &= \sum_{ijk} \epsilon_{ijk} A_{2i} A_{1j} A_{3k} \\ &= \sum_{ijk} \epsilon_{ijk} A_{1j} A_{2i} A_{3k} \\ &= -\sum_{jik} \epsilon_{jik} A_{1j} A_{2i} A_{3k} = -\det(A).\end{aligned}$$

Mediante un procedimiento análogo se puede probar también que

$$\det\left(T_{\alpha\beta V}(A)\right) = -\det(A),$$

por lo tanto

$$\det(T_{\alpha\beta}(A)) = \det(A).$$

□

A partir de este resultado y haciendo uso de la propiedad de linealidad de las transformaciones T se obtiene el siguiente corolario.

Corolario 1.1. *El conjunto de transformaciones T no altera los eigenvalores de la matriz de adyacencia.*

Demostración. Asumiendo que T representa a cualquier operador del tipo $T_{\alpha\beta}$, $T_{\alpha\beta H}$ o $T_{\alpha\beta V}$, y haciendo uso de la propiedad de linealidad, se tiene que

$$\text{eigen}(T(A)) = |T(A) - \lambda I| = |T(A - \lambda I)| = |A - \lambda I| = \text{eigen}(A).$$

□

5.4. Especificación de algoritmo

El resultado del corolario 1.1 nos lleva a que si A representa un DAG, entonces los eigenvalores de una matriz de adyacencia de un grafo equivalente deben ser también $\lambda = 1$. Se puede comprobar fácilmente que cualquier DAG tiene una representación equivalente en forma de matriz triangular superior, por ejemplo, el DAG representado por la matriz

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

tiene una representación equivalente en forma de matriz triangular superior dada por las transformaciones

$$T_{12} \circ T_{13}(A) = T_{13} \circ T_{23}(A) = T_{12} \circ T_{23}(A) = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

en consecuencia los eigenvalores de $A + I$ son $\lambda_i(A + I) = 1$. El algoritmo generador de DAG implementado en este proyecto genera matrices aleatorias y elimina conexiones en el grafo hasta que se cumple la condición $|\lambda_i| = 1$.

Data: tamaño grafo n
Result: Población de matrices correspondientes a un DAG
Generar una población de matrices de adyacencia a partir de matrices aleatorias con elementos {0,1};
foreach *Matriz de adyacencia* $A_i = A(G_i)$ **do**
  Obtener eigenvalores $\lambda_i = \lambda(A_i)$;
  if $|\lambda_i| = 1$ **then**
    A representa un DAG;
  else
    A **no** representa un DAG. Elimina conexión del grafo G_i y repite;
  end
end

Algorithm 1: Algoritmo generador de matrices de adyacencia DAG.

Este algoritmo se encuentra implementado en la función **GeneratePopulation**[*n*, *size*] que genera una población de tamaño *size* con un número *n* de atributos.

```
Map[MatrixForm, GeneratePopulation[4, 20]]
```

$$\left\{\begin{pmatrix} 0&0&0&0\\1&0&1&0\\0&0&0&0\\0&1&0&0 \end{pmatrix}, \begin{pmatrix} 0&1&0&0\\0&0&0&0\\0&1&0&1\\1&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&1&0&0\\0&0&0&0\\1&0&0&0\\0&0&1&0 \end{pmatrix}, \begin{pmatrix} 0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\1&0&1&0\\0&0&0&1\\1&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\1&0&1&0\\0&0&0&0\\0&0&1&0 \end{pmatrix},\right.$$

$$\begin{pmatrix} 0&0&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\0&0&0&0\\0&1&0&0\\1&1&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&1\\0&0&0&0\\1&1&0&1\\0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\1&0&0&1\\1&0&0&0\\1&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\0&0&0&0\\1&0&0&0\\0&1&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&0\\1&0&1&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\1&0&0&0\\1&1&0&0\\0&1&0&0 \end{pmatrix},$$

$$\left.\begin{pmatrix} 0&1&1&1\\0&0&0&1\\0&0&0&0\\0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&1&0&1\\0&0&0&0\\0&1&0&0\\0&0&1&0 \end{pmatrix}, \begin{pmatrix} 0&0&1&0\\0&0&1&0\\0&0&0&0\\1&0&1&0 \end{pmatrix}, \begin{pmatrix} 0&1&0&0\\0&0&0&1\\1&1&0&0\\0&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&1&0\\0&0&0&1\\0&0&0&0\\1&0&0&0 \end{pmatrix}, \begin{pmatrix} 0&1&0&0\\0&0&0&0\\0&0&0&0\\0&1&0&0 \end{pmatrix}, \begin{pmatrix} 0&0&0&0\\0&0&0&1\\1&0&0&0\\0&0&0&0 \end{pmatrix}\right\}$$

6. Métrica MDL

El principio de *longitud de mínima descripción* (minimum description length o MDL) afirma que el mejor modelo es aquel que permite la descripción más corta, en el contexto de la codificación de los datos y el modelo mismo [2]. La longitud de descripción el el número de bits requeridos para almacenar tal codificación.

La métrica se descompone en varios subcomponentes que dan lugar a la descripción completa del modelo.

6.1. Codificación de las variables

Primero se codifica el número de variables y el número posible de valores que puede tomar cada una. El número de posibles valores que puede tomar la variable aleatoria X_i se denota por $\|X_i\|$.

$$DL_{var} = \log n + \sum_{i=1}^{n} \log \|X_i\|$$

6.2. Codificación del DAG

Se puede describir el DAG guardando para cada variable desde X_1 hasta X_n el número de padres y una lista de padres. Dado que Π_i denota el conjunto de padres de X_i, entonces $|\Pi_i|$ se usa para denotar el número de padres de X_i.

$$DL_{dag} = \sum_{i=1}^{n} (1 + |\Pi_i|) \log n$$

6.3. Codificación de los parámetros

En la red bayesiana hay un parámetro por cada posible combinación de probabilidad condicional y posibles valores de las variables. Para esto se cuentan las posibles configuraciones $\|\Pi_i\|$ de los padres del nodo X_i y todos los $\|X_i\|$ posibles valores del nodo.

$$DL_{param} = \frac{1}{2} \log m \sum_{i=1}^{n} \|\Pi_i\| \left(\|X_i\| - 1\right)$$

6.4. Codificación de los datos

Los datos consisten en una secuencia m de realizaciones de $U = (X_1, X_2, \ldots, X_n)$. Usando la codificación de Shannon la contribución de los datos es:

$$DL_{data} = -\sum_{i=1}^{m} \log p(u_i)$$

6.5. MDL

La puntuación MDL se define como la longitud total de descripción del modelo y los datos.

$$MDL = DL_{var} + DL_{dag} + DL_{param} + DL_{data}$$

El cálculo de la puntuación MDL se encuentra implementado en la función **MDLScore**[*dag*, *dataset*], donde *dag* es una lista de conexiones del grafo y *dataset* es la lista de instancias. Nótese que el valor del MDL de un modelo con estructura no trivial:

```
N@MDLScore[{1 ↦ 2, 1 ↦ 3, 2 ↦ 3, 3 ↦ 4}, irisDataset]
485.299
```

es menor que un modelo vacío:

```
N@MDLScore[{}, irisDataset]
672.565
```

7. Evaluación de aptitud

Existen múltiples esquemas para particionar los datos en conjuntos de entrenamiento y prueba con diferentes propiedades. En este trabajo se utilizaron tres:

- Evaluación con mismos datos de entrenamiento y prueba.
- Evaluación con minibatches.
- Evaluación con validación cruzada.

La evaluación con mismos datos de entrenamiento y prueba es lo más rápido pero puede generar modelos que sobreajusten los datos y que tengan poca capacidad predictiva. La evaluación con minibatches consiste en entrenar y evaluar con subconjuntos aleatorios de la base de datos. Esto es útil si la base de datos es demasiado grande. Por último el esquema de validación cruzada consiste en hacer múltiples particiones a los datos como se muestra en la figura 6 y entrenar seleccionando las particiones correspondientes a cada iteración.

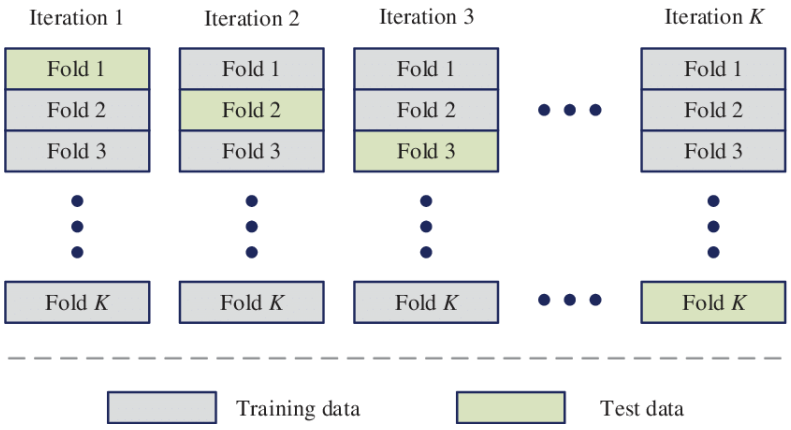


Figura 6: Diagrama del algoritmo de validación cruzada.

Esto se implementó en las funciones **EvaluateGenome**, **EvaluateGenomeWithMinibatch** y **EvaluateGenomeWithKFolds**. Un ejemplo de uso es:

8. Optimización genética

8.1. Selección de población

La selección se hace de manera aleatoria y proporcional, dando preferencia a los individuos más aptos. El algoritmo de selección ordena los individuos según

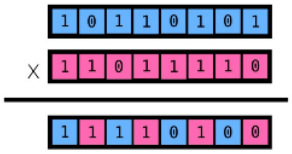
```
EvaluateGenome[irisDataset, {{{0,0,0,0},{1,0,0,0},{0,0,0,0},{0,1,1,0}}, {{0,1,0,0},{0,0,0,0},{0,0,0,0},{1,0,1,0}}, {{0,0,0,0},{0,0,1,0},{0,0,0,0},{0,1,0,0}}, {{0,0,0,1},{1,0,1,0},{0,0,0,1},{0,0,0,0}}}, MDLScore]

{⟨|Genome → {{0, 0, 0, 0}, {1, 0, 0, 0}, {0, 0, 0, 0}, {0, 1, 1, 0}}, Score → 524.761|⟩,
 ⟨|Genome → {{0, 1, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {1, 0, 1, 0}}, Score → 489.269|⟩,
 ⟨|Genome → {{0, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 0}, {0, 1, 0, 0}}, Score → 632.771|⟩,
 ⟨|Genome → {{0, 0, 0, 1}, {1, 0, 1, 0}, {0, 0, 0, 1}, {0, 0, 0, 0}}, Score → 553.659|⟩}
```

su aptitud escoge un índice aleatorio con mayor peso en los individuos más aptos.

ProportionateSelection[*population*, *n*, *selectionCriteria*] realiza la selección a partir de la población evaluada *population*, tomando *n* individuos con el criterio de selección *selectionCriteria* que puede tomar los valores "LessIsBetter.º "MoreIsBetter".

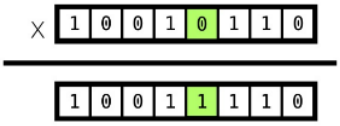
8.2. Operador de cruza

La cruza se realiza de manera aleatoria, seleccionando un valor de cualquiera de ambos padres haciendo un muestreo aleatorio. **PopulationCrossover**[*population*, *childrenPerCouple*] realiza la cruza emparejando aleatoriamente pares de individuos dentro de la población produciendo *childrenPerCouple* hijos por cada par.



8.3. Mutación

La mutación se realiza cambiando un solo elemento del genoma en una posición aleatoria. **PopulationMutate**[*population*,*probability*] aplica mutationes aleatoriamente con una probabilidad *probability* sobre cada individuo de la población. Sólo un alelo es modificado por cada mutación.



9. Resultados

Como resultado se obtiene una interfaz de usuario que permite encontrar la mejor estructura. **GenerateInitialPopulation**[*size*, *dataset*, *opts*] genera una población inicial de una cantidad *size* de individuos y evalúa su rendimiento en la base de datos *dataset*. Sus posibles argumentos opcionales son:

- *ScoringFunction* que puede tomar los valores *MDL* para evaluar usando la métrica MDL y *CorrectlyClassified* para evaluar usando la métrica de porcentaje de instancias correctamente evaluadas.
- *TestMethod* puede tomar los valores *UseTrainingSet* para evaluar usando el mismo conjunto de datos de entrenamiento, *UseMiniBatch* para evaluar por medio de minibatches de tamaño especificado en la opción *MiniBatchSize*, y *KFoldCrossValidation* para evaluar usando validación cruzada de n pliegues especificados en la opción Folds.
- *MiniBatchSize* es el tamaño del minibatch. Por defecto no está especificado, por lo que es necesario introducir esta opción explicitamente si se especifica TestMethod → UseMiniBatch.
- *Folds* es el número de pliegues en la validación cruzada. Por defecto tiene el valor de 10.

Generation[*dataset*, *population*, *survivors*, *mutationProb*, opts] realiza una generación del algoritmo genético con la población *population*, el número de sobrevivientes *survivors*, y la probabilidad de mutación *mutationProb*. Sus posibles argumentos opcionales son:

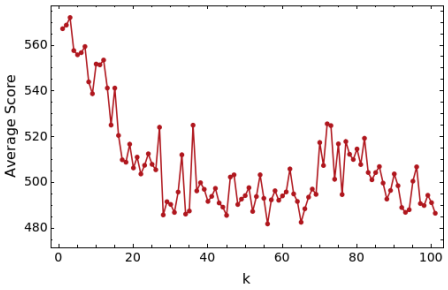
- *ScoringFunction*, *TestMethod*, *MiniBatchSize* y *Folds* usados en la función **GenerateInitialPopulation**, y
- *ScoreType* que especifica la preferencia del algoritmo de selección proporcional, y puede tomar los valores *LessIsBetter* para dar preferencia a puntuaciones con menor valor y *MoreIsBetter* para dar preferencia a puntuaciones con mayor valor.

GeneticOptimize[*dataset, population, survivors, mutationProb, generations*, opts] realiza una cantidad *generations* de iteraciones del algoritmo evolutivo.

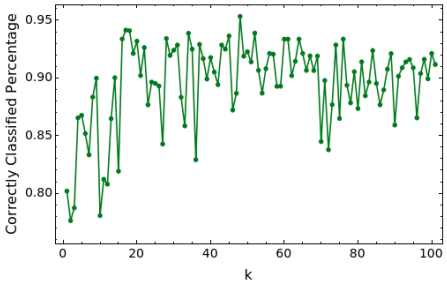
También se implementaron funciones auxiliares para visualización de resultados:

- **SelectBest**[*population, n, scoreType*] selecciona los *n* individuos más aptos de la población de acuerdo al criterio *scoreType*.
- **PlotBayesNet**[*net, attributes*] grafica la red bayesiana.
- **PlotGeneration**[*generation, attributes, index*] grafica los individuos de una generación en la iteración especificada en *index*.
- **PopulationCorrectlyClassifiedPercentage**[*population, testData*] calcula el porcentaje de aciertos promedio de una población sobre una base de datos de prueba.

Las figuras 7, 8, 9, 10, 11 y 12 muestran los resultados del proceso de entrenamiento para las bases de datos Iris, Car, Bank, Adult, Servo y Tic-Tac-Toe obtenidas del UCI Dataset.

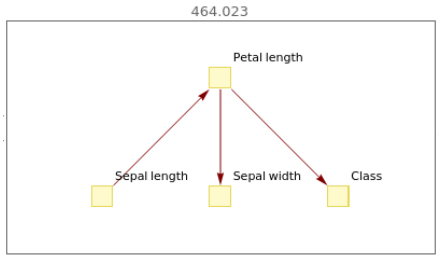


(a) Puntuación MDL por generación.



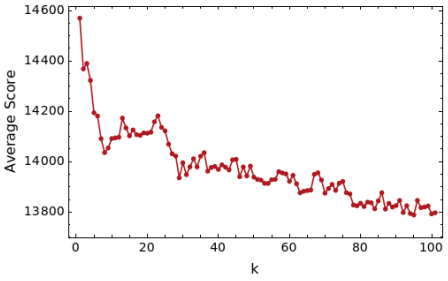
(b) Porcentaje de instancias correctamente clasificadas por generación.

(c) Primera generación.

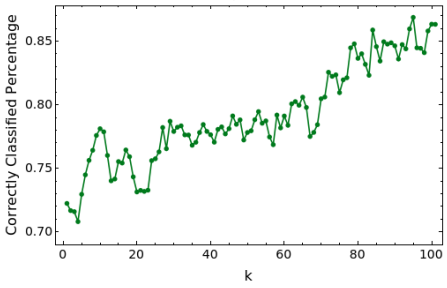


(d) Estructura óptima alcanzada en la generación 100.

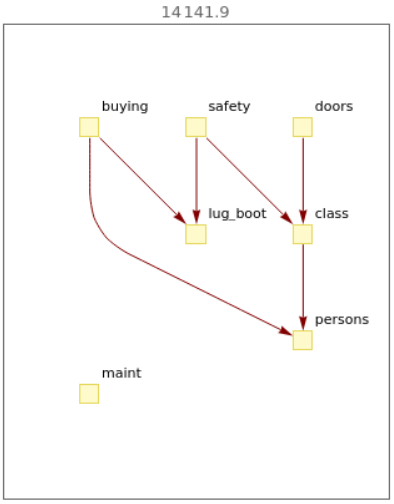
Figura 7: Resultados para la base de datos Iris.



(a) Puntuación MDL por generación.



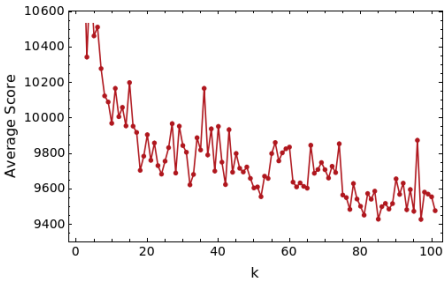
(b) Porcentaje de instancias correctamente clasificadas por generación.



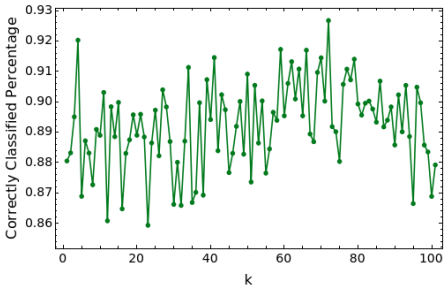
(c) Primera generación.

(d) Estructura óptima alcanzada en la generación 100.

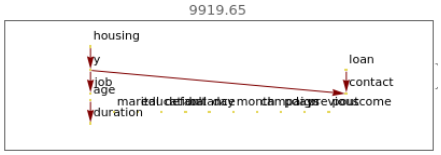
Figura 8: Resultados para la base de datos Car.



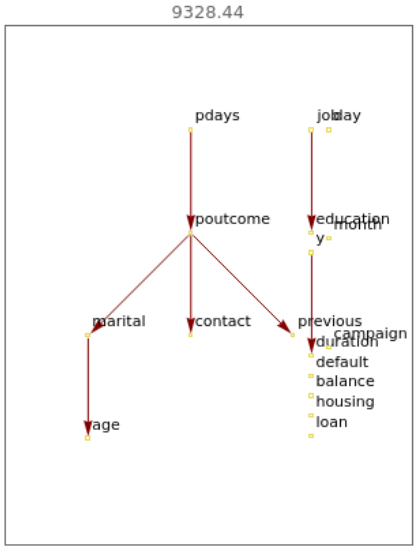
(a) Puntuación MDL por generación.



(b) Porcentaje de instancias correctamente clasificadas por generación.

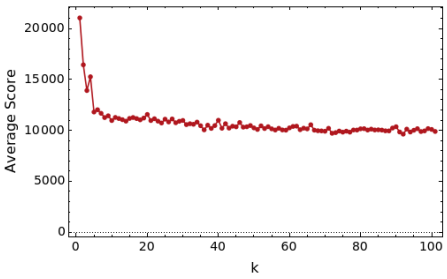


(c) Primera generación.

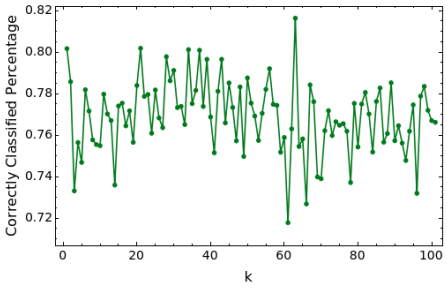


(d) Estructura óptima alcanzada en la generación 100.

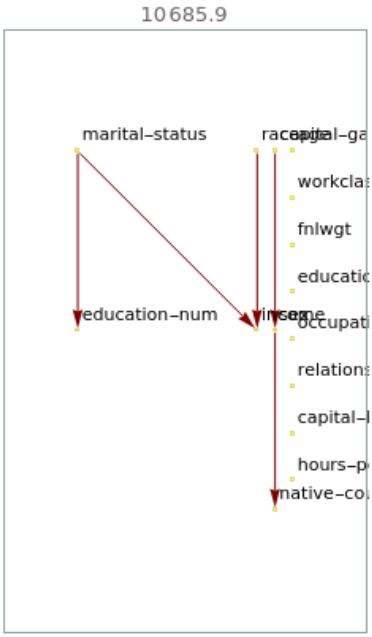
Figura 9: Resultados para la base de datos Bank.



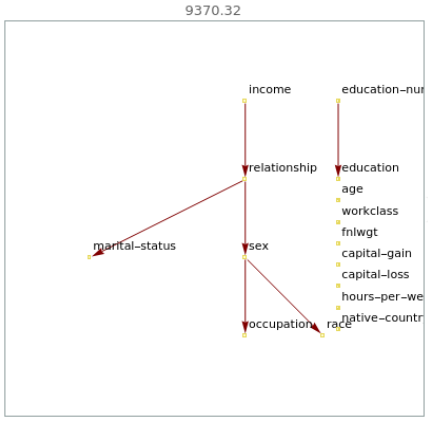
(a) Puntuación MDL por generación.



(b) Porcentaje de instancias correctamente clasificadas por generación.

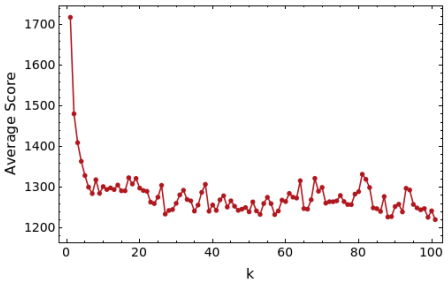


(c) Primera generación.

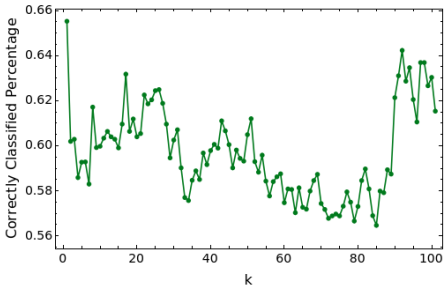


(d) Estructura óptima alcanzada en la generación 73.

Figura 10: Resultados para la base de datos Adult.

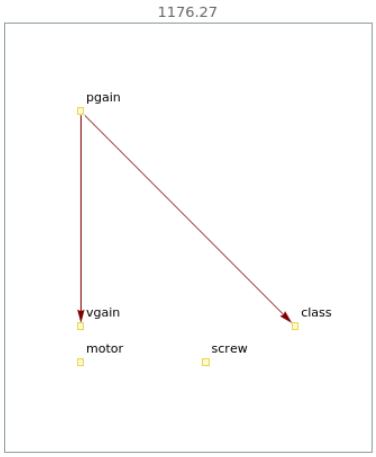


(a) Puntuación MDL por generación.



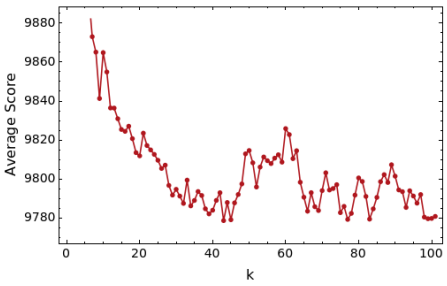
(b) Porcentaje de instancias correctamente clasificadas por generación.

(c) Primera generación.

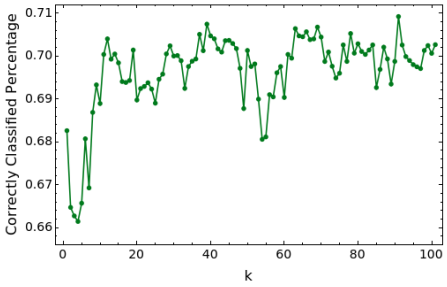


(d) Estructura óptima alcanzada en la generación 100.

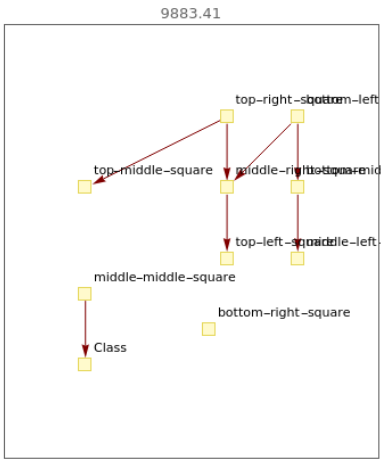
Figura 11: Resultados para la base de datos Servo.



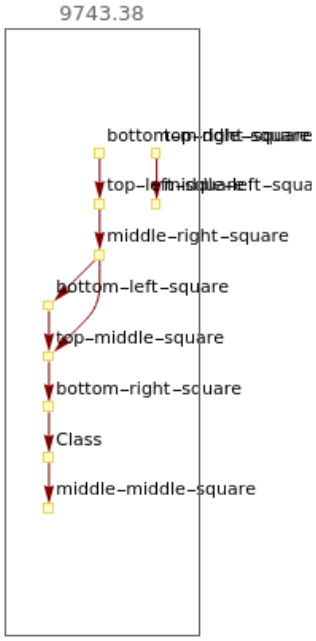
(a) Puntuación MDL por generación.



(b) Porcentaje de instancias correctamente clasificadas por generación.



(c) Primera generación.



(d) Estructura óptima alcanzada en la generación 100.

Figura 12: Resultados para la base de datos Tic-Tac-Toe.

Referencias

[1] Eugene Charniak. Bayesian networks without tears.

[2] Nicholas D. Levine. Using minimum description length for discretization classification of data modeled by bayesian networks.

[3] Brendan D. McKay, Frederique E. Oggier, Gordon F. Royle, N. J. A. Sloane, Ian M. Wanless, and Herbert S. Wilf. Acyclic digraphs and eigenvalues of (0,1)-matrices.

[4] Nicandro Cruz Ramírez. Building bayesian networks from data: a constraint-based approach.

[5] R. W. Robinson. Counting unlabeled acyclic digraphs. In *Lecture Notes in Mathematics*, pages 28–43. Springer Berlin Heidelberg, 1977.