2023

# REDES SEMÁNTICAS

NICOLAS CAO PARGA
MARTIN ARES ALVAREZ

# Redes Semánticas: Concepto y aplicaciones

La Red Semántica es una herramienta de representación muy útil de adquirir conocimientos de forma organizada y estructurada. Estas redes se basan en nodos que representan conceptos y círculos que representan relaciones entre estos conceptos. Estas redes se utilizan en diversos campos como la inteligencia artificial, psicología cognitiva y lingüística. A continuación, se profundizará sobre este tema y se explicará cómo se aplican las redes semánticas en diferentes entornos.

## Conceptos claves en las redes semánticas

En una red semántica, los nodos representan conceptos y arcos representan la relación entre ellos. Los nodos son entidades específicas que pueden ser, por ejemplo, un objeto físico o abstracto como una idea. Los círculos representan la relación existente entre dos nodos. Algunas de las condiciones más comunes son: Causalidad, totalidad, igualdad, etc.

La red semántica funciona identificando conceptos clave y la relación entre ellos. Un concepto clave es un elemento necesario para representar la relación en la Red Semántica y las relaciones son las conexiones entre los elementos.

Existen dos relaciones más utilizadas en las redes semánticas las cuales son:

- ES-UN: Es un enlace que se emplea para representar el hecho de que un elemento es miembro de una clase de elementos que tienen un conjunto de propiedades distintivos, en común. Un nodo que representa una ilustración de una clase es una instancia (ejemplo) de la clase. Los conceptos de una clase y de un enlace ES-UN se utilizan también para representar situaciones, acciones y eventos.

- ES-SUBCONJUNTO: “Las redes semánticas son una representación gráfica de saber sobre objetos y sus relaciones”. El razonamiento con redes semánticas es directo puesto que las asociaciones se pueden hacer simplemente rastreando los enlaces en el sistema, a este mecanismo se le llama propagación de la activación. Desafortunadamente, ninguna regla semántica rigurosa guía tal razonamiento. La interpretación de las estructuras de la red depende solamente del programa que las manipula, es decir, que no existe convención del significado, por esta razón, las inferencias que se derivan de la manipulación de la red no son necesariamente válidas.

## Aplicaciones de las redes semánticas

Las redes semánticas son herramientas muy útiles en diferentes campos como la inteligencia artificial, la psicología cognitiva y la lingüística. En la inteligencia artificial, se utilizan para representar el conocimiento de una máquina sobre un tema determinado, mientras que en la psicología cognitiva se utilizan para modelar cómo las personas organizan y representan el conocimiento en su mente. En el campo de la lingüística, se utilizan para representar las relaciones entre las palabras de un idioma. Además, las redes semánticas son utilizadas en las ontologías, sistemas de representación del conocimiento que utilizan redes semánticas para modelar y organizar el conocimiento en una determinada área.

## Trabajo: Aplicación Red semántica al ámbito universitario

Las redes semánticas son una herramienta valiosa para representar y organizar el conocimiento en el contexto universitario. Las redes semánticas se pueden utilizar para representar la estructura de un curso, las relaciones entre los diferentes conceptos que se enseñan, y para ayudar a los estudiantes a entender y memorizar el material. En el contexto

universitario, una red semántica puede ser utilizada para representar los diferentes temas que se abordan en un curso. Cada tema se convierte en un nodo en la red semántica y las relaciones entre los diferentes temas se representan mediante arcos que conectan los nodos correspondientes. Las relaciones entre los temas pueden ser de diferentes tipos, como causales, de influencia, jerárquicas, entre otras. Un ejemplo concreto de cómo se pueden utilizar las redes semánticas en el contexto universitario es en la muestra de un trabajo a un grupo de alumnos.

Para representar esta situación en una red semántica, se pueden identificar los siguientes conceptos clave:

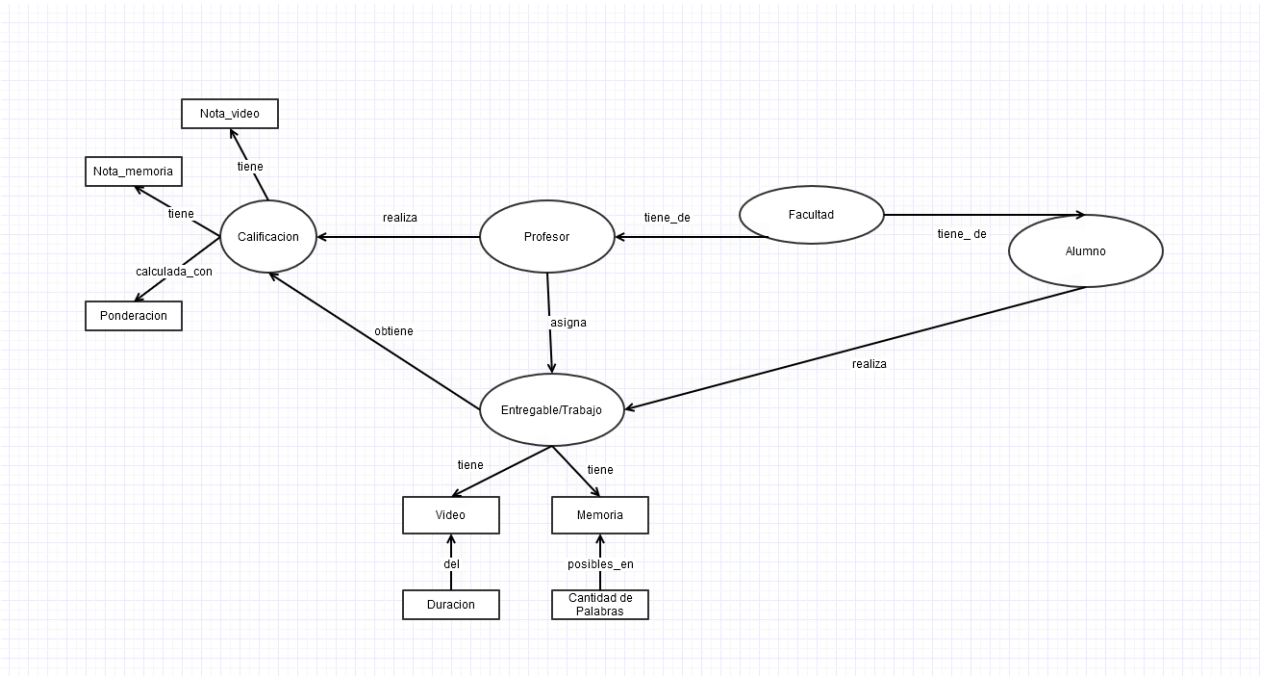
1- Profe
2- Alumnos
3- Trabajo
4- Memoria
5- Video
6- Calificación

Las relaciones entre estos conceptos pueden ser representadas como sigue:

1- El profesor asigna el trabajo a los alumnos.
2- El trabajo consta de dos entregas: una memoria y un vídeo.
3- Cada entrega se califica por separado.
4- Las calificaciones tienen un peso diferente.

En esta red semántica, los conceptos clave son los nodos y las relaciones entre ellos son los arcos. Al representar la situación de esta manera, se puede visualizar de manera clara la estructura del trabajo y las relaciones entre los diferentes conceptos. Además de representar la estructura de un curso o la adornar de un trabajo, las redes semánticas también se pueden utilizar para ayudar a los estudiantes a entender y memorizar el material. Por ejemplo, se puede utilizar una red semántica para representar los diferentes conceptos que se abordan en una clase de biología y las relaciones entre ellos. Al visualizar la red semántica, los estudiantes pueden comprender mejor cómo los diferentes conceptos se relacionan entre sí y cómo están organizados.

El UML o los diagramas de entidad relación seria asi el siguiente para el problema formulado:




El código de AgentSpeak sería el siguiente:

```
entregables(video).
entregables(memoria).

alumno(alumno1).
alumno(alumno2).

profesor(moreno).

insertar(X,[],[X]).
insertar(X, Lista, [X|Lista]).

por_entregable([Entregable], Alumno, L1, L3):-
   insertar(trabajo_entregado(Alumno, Entregable), L1, L3).
por_entregable([Entregable|Resto], Alumno, L1, L3):-
   insertar(trabajo_entregado(Alumno, Entregable), L1, L2) &
```

```
        por_entregable(Resto, Alumno, L2, L3).

por_alumno([Alumno], Entregables, L1, L3):- por_entregable(Entregables, Alumno, L1, L3).
por_alumno([Alumno|Resto], Entregables, L1, L3):-
   por_entregable(Entregables, Alumno, L1, L2) &
        por_alumno(Resto, Entregables, L2, L3).

despachar_entregables(Alumnos, Entregable, L1, L2):-
   por_alumno(Alumnos, Entregable, L1, L2).

mandar_hacer_un_trabajo(Profesor, L1, L2):-
   profesor(Profesor) &
   .findall(Alumno, alumno(Alumno), Alumnos) &
   .findall(Entregable, entregables(Entregable), Entregables) &
   despachar_entregables(Alumnos, Entregables, L1, L2).

mandar_hacer_un_trabajo(Profesor, L1, L2):-
   .print("mandar_hacer_un_trabajo NOK").

calificacion_random1(X):-
   Tope = 10 &
   .random(Val, Tope) &
   X = Val * 10 * 0.6.

calificacion_random2(X):-
   Tope = 10 &
   .random(Val, Tope) &
   X = Val * 10 * 0.4.

puntuar([Trabajo],Alumno, L1, L2):-
   calificacion_random1(Nota) &
```

```
    insertar(calificacion(Alumno, Trabajo, Nota), L1, L2) &
            .print(L2).

puntuar([Trabajo|Resto],Alumno, L1, L3):-
    calificacion_random2(Nota) &
    insertar(calificacion(Alumno,Trabajo, Nota), L1, L2) &
    puntuar(Resto, Alumno, L2, L3).

calificar_alumnos([Alumno],Trabajos, L1, L2):-
    .findall(X, trabajo_entregado(Alumno, X), Trabajos) &
    puntuar(Trabajos, Alumno, L1, L2).

calificar_alumnos([Alumno|Resto],Trabajos, L1, L3):-
    .findall(X, trabajo_entregado(Alumno, X), Trabajos) &
    puntuar(Trabajos, Alumno, L1, L2) &
    calificar_alumnos(Resto, Trabajos, L2, L3).

calificar_trabajos(Profesor, L1, L2):-
    profesor(Profesor) &
    .findall(Alumno, alumno(Alumno), Alumnos) &
    calificar_alumnos(Alumnos,Trabajos, L1, L2).

calificar_trabajos(Profesor, L1, L2):- .print(" calificar_trabajos NOK").

sumar([H], Acc, R):-
    R = Acc + H.
sumar([H|T], Acc, R):-
    Aux = Acc + H &
    sumar(T, Aux, R).

calcular_notas_finales([Alumno]):-
```

```
    .findall(X, calificacion(Alumno,_,X), Calificaciones) &
    sumar(Calificaciones, 0, R) &
    .print("Nota de ", Alumno, " es ", R).

calcular_notas_finales([Alumno|Resto]):-
    .findall(X, calificacion(Alumno,_,X), Calificaciones) &
    .print(Calificaciones) &
    sumar(Calificaciones, 0, R) &
    .print("Nota de ", Alumno, " es ", R) &
    calcular_notas_finales(Resto).

+!almacenar_beliefs(Beliefs) <-
    for ( .member(M,Beliefs) ) {
        +M;
    }.

!start.
+!start : true <-
    ?mandar_hacer_un_trabajo(moreno, [], Entregables);
    !almacenar_beliefs(Entregables);
    ?calificar_trabajos(moreno, [], Resultados);
    !almacenar_beliefs(Resultados);
    .findall(Alumno, alumno(Alumno), Alumnos);
    ?calcular_notas_finales(Alumnos).
```

Este código implementa un programa Prolog que simula el proceso de calificación de trabajos entregados por alumnos. A continuación, se describe en detalle la funcionalidad de cada uno de los predicados y reglas del programa:

entregables(video) y entregables(memoria): estos son hechos que se utilizan para definir los tipos de trabajos que pueden ser entregados por los alumnos.

alumno(nico) y alumno(martin): estos son hechos que se utilizan para definir los alumnos que participan en la actividad.

profesor(moreno): este es un hecho que se utiliza para definir el profesor encargado de calificar los trabajos.

insertar(X,[],[X]) y insertar(X, Lista, [X|Lista]): estos son predicados auxiliares que se utilizan para insertar elementos en una lista.

mandar_hacer_un_trabajo(Profesor, L1, L2): este predicado se encarga de simular el proceso de entrega de trabajos por parte de los alumnos. Recibe como argumentos el nombre del profesor encargado (Profesor), una lista vacía L1 que se utiliza para almacenar los trabajos entregados por los alumnos y otra lista vacía L2 que se utiliza como resultado. Primero se verifica que el Profesor sea un profesor válido. Luego, se obtienen la lista de Alumnos y la lista de Entregables mediante el uso de findall. Finalmente, se llama al predicado despachar_entregables pasando como argumentos Alumnos, Entregables, L1 y L2.

despachar_entregables(Alumnos, Entregable, L1, L2): este predicado se encarga de despachar los trabajos entregados por los alumnos. Recibe como argumentos la lista de Alumnos que participan en la actividad, la lista de Entregables disponibles, una lista vacía L1 que se utiliza para almacenar los trabajos entregados por los alumnos y otra lista vacía L2 que se utiliza como resultado. Primero se llama al predicado por_alumno pasando como argumentos Alumnos, Entregables, L1 y L2.

por_alumno([Alumno], Entregables, L1, L3) y por_alumno([Alumno|Resto], Entregables, L1, L3): estos predicados se encargan de procesar los trabajos entregados por un alumno específico. El primero se utiliza cuando sólo queda un alumno por procesar, mientras que el segundo se utiliza para procesar el resto de los alumnos. Se llama al predicado por_entregable pasando como argumentos Entregables, Alumno, L1 y L2. Luego, se llama a por_alumno recursivamente pasando como argumentos Resto, Entregables, L2 y L3.

por_entregable([Entregable], Alumno, L1, L3) y por_entregable([Entregable|Resto], Alumno, L1, L3): estos predicados se encargan de procesar un trabajo específico entregado por un alumno. El primero se utiliza cuando sólo queda un trabajo por procesar, mientras que el segundo se utiliza cuando hay varios trabajos por procesar.

La regla calificar_alumnos recibe una lista de alumnos y una lista de trabajos. Primero, utiliza findall para obtener los trabajos entregados por el alumno actual y los almacena en la variable Trabajos. Luego, llama a la regla puntuar con los trabajos entregados por el alumno actual y la variable Alumno. La regla puntuar recibe una lista de trabajos, un alumno y dos listas. En la regla puntuar, primero se utiliza la regla calificacion_random1 para obtener una nota aleatoria para el primer trabajo y la regla calificacion_random2 para obtener notas aleatorias para los trabajos restantes. Luego, se utiliza insertar para agregar una nueva calificación a la lista de calificaciones y se llama recursivamente a puntuar con la lista de trabajos restantes.

La regla calificar_trabajos recibe un profesor y dos listas. Primero, utiliza findall para obtener una lista de todos los alumnos y los almacena en la variable Alumnos. Luego, llama a la regla calificar_alumnos con la lista de alumnos y la lista de trabajos entregados.

La regla calcular_notas_finales recibe una lista de alumnos. Primero, utiliza findall para obtener una lista de todas las calificaciones del alumno actual y la almacena en la variable Calificaciones. Luego, utiliza la regla sumar para sumar todas las calificaciones y almacenar el resultado en la variable R. Por último, muestra por pantalla la nota final del alumno actual.

La última parte del código es el planificador !start. Cuando se llama a !start, se utiliza la regla mandar_hacer_un_trabajo para generar los entregables y se almacenan en la variable Entregables. Luego, se utiliza la regla calificar_trabajos para calificar los trabajos entregados y se almacenan los resultados en la variable Resultados. Después, se utiliza findall para obtener una lista de todos los alumnos y se llama a calcular_notas_finales con la lista de alumnos. Por último, se utiliza la regla !almacenar_beliefs para almacenar todos los beliefs generados en la base de conocimientos.