Diseño de Agentes Inteligentes Difusos para Marketing Web

Entrega Final

Licenciatura en Informática

Facultad de Informática

Universidad Nacional de La Plata

Alumnos:
Carolina Vidaurreta 1617/6
Mónica Donajger 1833/3

Director:
Dra. Clara Smith

Mayo de 2005

Agradecimientos

Queremos agradecer muy especialmente a nuestros padres por brindarnos la posibilidad de realizar una carrera universitaria. Gracias por no haber dudado en enviarnos a estudiar a otra ciudad a pesar de que esto les representaba separarse de sus hijas y un gran esfuerzo económico.

Muchísimas gracias a Clara Smith, nuestra directora, que nos guió y aconsejó en el transcurso de nuestra tesis.

Gracias a todos los que nos ayudaron de una u otra manera para que este trabajo se viera concluido. Mil gracias a los que nos brindaron conocimientos y consejos para el desarrollo de este trabajo de grado.

Tampoco podemos de dejar de mencionar a una personita muy especial, llamada Milena, que con sus "gracias" amenizaba las horas de estudio.

Gabriele: Gracias por tu ayuda, comprensión y predisposición constante. Infinitas gracias por acompañarme siempre y transmitirme confianza. Pero principalmente gracias por estar en todo momento a mi lado y ayudarme a conseguir juntos este logro.

Claudio: Hemos estado juntos desde los tiempos de la escuela secundaria y por lo tanto toda mi carrera fue a tu lado. Los primeros años como novio y como compañero de estudios, luego como esposo y ahora también como padre de nuestra hijita. Gracias por toda la paciencia, apoyo y estímulo que me brindaste y me seguís brindando.

Mónica Donajger

Carolina Vidaurreta

La Plata, 16 de Mayo de 2005

Índice General

Capítulo 1

Introducción

En la última década se han producido fenómenos inéditos en la historia de la humanidad debido a la explosión de los avances tecnológicos. Uno de ellos es Internet y su conversión en un nuevo medio para la comunicación social y el mercadeo. Se dice que es el medio de comunicación y venta con el mayor potencial de crecimiento y efectividad y al más bajo costo: un excelente canal de comunicación humana. Esto lo convierte en una plataforma publicitaria única.

Sin embargo, sabemos que la publicidad en Internet ha sido cuestionada por varios factores, por ejemplo, que no atrae a los navegantes, que llega a muy pocas personas, que comparada con la publicidad en televisión es muy costosa.

Los problemas de la publicidad en Internet son varios, entre ellos está el hecho de que por lo general quien navega busca información específica y sin perder demasiado tiempo en ello. El navegante busca algo, pero generalmente no busca anuncios. El navegante busca información concreta, busca contactos con gente, y la publicidad por lo general le molesta.

Uno de los secretos para sacarle partido a la publicidad en Internet es presentarla en el lugar en el que se halla el potencial cliente. Un buen publicista, obviamente, sabe localizarlo para colocarle allí su mensaje. El marketing verdaderamente orientado puede ser útil e informativo para el destinatario. Dos decisiones importantes para el anunciante son qué productos ofrecerle y de qué modo hacerlo.

Esta problemática hace surgir la posibilidad de que los anunciantes de cierto sitio de Internet puedan determinar si un visitante pertenece a su mercado meta y en qué grado.

Teniendo en cuenta los problemas que aquejan a los usuarios al navegar en la Web y los problemas que aquejan a los anunciantes en su intento de llegar a sus potenciales clientes sin desperdiciar anuncios, hemos diseñado una política que pretende ayudar a los anunciantes a tomar decisiones respecto los visitantes de sitios Web con el objetivo de que la publicidad resulte beneficiosa tanto para el anunciante como para el visitante, porque no verá publicidad irrelevante.

En otro orden de cosas, la lógica difusa contribuye con su capacidad formal para representar la conceptualización humana.

Mostraremos como, usando lógica difusa, podemos determinar cuáles son los anuncios que mejor se adecuan a un determinado visitante del sitio a partir del conocimiento que tengamos de aquél. De esta manera, los anunciantes pueden contar con la posibilidad de mostrar anuncios con muchas más chances de despertar interés.

Elegimos lógica difusa porque nos interesa que nuestro sistema modele la toma de decisiones de modo similar a como lo hace la mente humana. Ésta está capacitada para formular planes, tomar decisiones y reconocer conceptos compatibles con altos niveles de vaguedad y ambigüedad. Para capturar esta forma de razonar el modelado tradicional booleano es muchas veces insuficiente. Utilizando lógica difusa logramos de algún modo capturar formalmente, formas cotidianas de razonamiento humano.

1.1 Esquema de la tesis

La tesis está dividida en capítulos. En el capítulo 2 se trata una serie de conceptos necesarios para la comprensión general de trabajo. En la sección 2.1 se exponen nociones de Lógica Difusa, haciendo hincapié en el proceso de Inferenciamiento Difuso. En la sección 2.2 se explican algunos conceptos de Marketing y Publicidad en la Web. En la sección 2.3 se describe el concepto de agentes.

Introducimos nuestra propuesta: "Selector de publicidad para la Web" en el capítulo 3. En la sección 3.1 se detalla su definición arquitectónica; su división en capas, las interrelaciones entre ellas, los agentes que funcionan en su interior y las relaciones entre tales agentes. En la sección 3.2, describimos las páginas web que componen la interfaz del usuario común y del usuario administrador (anunciantes).

En el Capítulo 4 continuamos la presentación de nuestra propuesta detallando "El proceso de selección de la publicidad". En la sección 4.1 se trata el ingreso del usuario al sitio. La sección 4.2 describe cada una de las componentes que intervienen en la selección de la publicidad junto con el rol que cumple cada una dentro del sistema. La sección 4.3 explica el proceso de muestra de la publicidad.

En el Capítulo 5 hemos construido un "ejemplo de trabajo" para poder apreciar y comprobar la funcionalidad del selector de publicidad propuesto. La idea principal es integrar todos los conceptos tratados en la tesis en un caso práctico que creemos puede ayudar a lograr una mejor comprensión de lo desarrollado en los capítulos precedentes.

En el Capítulo 6 describimos las conclusiones de nuestro trabajo, así como perspectivas futuras de nuestro selector de publicidad.

Finalmente, en el Apéndice, incluimos la reseña técnica de clases que componen el selector de publicidad y que forman parte de la implementación computacional lograda.

Capítulo 2

Conceptos preliminares

2.1 - Lógica Difusa

2.1.1- Origen y conceptos

La mayoría de los modos con que las personas nos referimos a los fenómenos de cada día son imprecisos, es decir, tienen implícito un cierto grado de ambigüedad y vaguedad. El hombre, en la búsqueda de precisión, ha intentado ajustar descripciones del mundo real a modelos matemáticos. Lo que se busca a través de la lógica difusa es describir y formalizar la realidad empleando modelos flexibles que interpreten las leyes que rigen el comportamiento humano y las relaciones entre los hombres [LMP98].

La lógica difusa nació en los ´60 como la "lógica del razonamiento aproximado". Es un superconjunto de la lógica convencional Booleana que ha sido extendida para manejar el concepto de verdad parcial, es decir, valores de verdad entre completamente verdadero y completamente falso [KHJ93].

Por ejemplo, supongamos que estamos hablando de la "mediana edad". Al escuchar este término, mentalmente lo asociamos a un determinado grupo de personas. Para modelizarlo, la lógica difusa utiliza una función de pertenencia [0,1] entre un elemento del dominio (en nuestro caso será el intervalo 0..120 años) y un determinado conjunto (para nosotros la mediana edad). Si partimos de la base de que la mediana edad son los 45 años la función de pertenencia entre los años y la mediana edad será máxima y valdrá 1. Sin embargo no podemos descartar a las personas de 35 o 55 años, ni tampoco a los menores de 35 años y los mayores de 55, que se encuentran próximos a los 45 años.

Entonces, la idea principal en lógica difusa es crear una relación matemática entre un elemento y un determinado conjunto de valores con el fin de que una computadora sea capaz de realizar automáticamente una apreciación similar a como lo hacemos nosotros, los humanos.

Gracias a la función de pertenencia, automáticamente puede interpretarse cuando un elemento pertenece o no a un conjunto difuso y lo que es más importante, puede evaluarse si ese grado de pertenencia es, por ejemplo, elevado (cercano a 1) o despreciable (cercano a 0).

La palabra "difuso" es un adjetivo técnico. Los fenómenos o procesos pueden ser difusos pero la teoría en sí es precisa y los sistemas computacionales difusos también se definen, claramente, de una manera precisa. El adjetivo se debe a que los valores de verdad utilizados en ellas tienen, por lo general, una connotación de *vaguedad*.

2.1.2- Conjuntos difusos y operaciones sobre conjuntos difusos

Un conjunto clásico (discreto) es una colección de objetos. Los elementos del universo de discurso se clasifican en dos grupos: miembros y no miembros. Sea U un universo de discurso. La función de pertenencia $\mu_A(x)$ de un conjunto discreto A en U toma sus valores en {0,1} y es definida de forma tal que $\mu_A(x)=1$ si x es un miembro de A y 0 en otro caso [NFS]. Esto es,

$$\mu_A(x) = \begin{cases} 1 & \text{sí y solo sí } x \in A \\ 0 & \text{sí y solo sí } x \notin A \end{cases}$$

Notemos que el límite del conjunto A es "rígido", bien definido y realiza una clasificación ($x \in A$ o $x \notin A$), y que el universo de discurso U es un conjunto discreto.

Los conjuntos difusos pueden ser vistos como una extensión y generalización de los conceptos básicos de conjuntos discretos. Un conjunto difuso introduce "vaguedad" desdibujando, de algún modo, los límites bien definidos. Así, la transición entre pertenencia completa y no-pertenencia es gradual antes que abrupta.

Definición: Un conjunto difuso $\bar{A}$ en el universo de discurso U puede ser definido como un conjunto de pares ordenados,

$$\bar{A} = \{ (x, \mu_{\bar{A}}(x)) \mid x \in U\},$$

Donde $\mu_{\bar{A}}$ se denomina "función de pertenencia" de $\bar{A}$, siendo $\mu_{\bar{A}}(x)$ el grado de pertenencia de x en $\bar{A}$. La función de pertenencia $\mu_{\bar{A}}$ aplica U en un espacio de pertenencia M, esto es, $\mu_{\bar{A}} : U \rightarrow M$. Cuando $M = \{0,1\}$, el conjunto A es no difuso y μ_A es la función característica del conjunto discreto A. Para conjuntos difusos, el rango de la función de pertenencia (M) es un subconjunto de los números reales no negativos cuyo supremo es finito. Usualmente M es un conjunto en el intervalo [0,1].

Ejemplo:

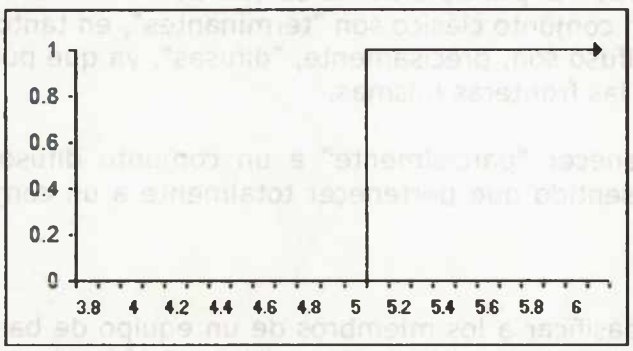
Sea U el conjunto de los números reales y sea A el conjunto discreto constituido por los "números reales mayores o iguales a 5"; entonces tenemos

$$A = \{ (x, \mu_A(x)) \mid x \in U\},$$

donde la función característica es

$$\mu_A(x) = \begin{cases} 0, & x < 5 \\ 1, & x >= 5, \end{cases}$$

que graficamos como sigue:



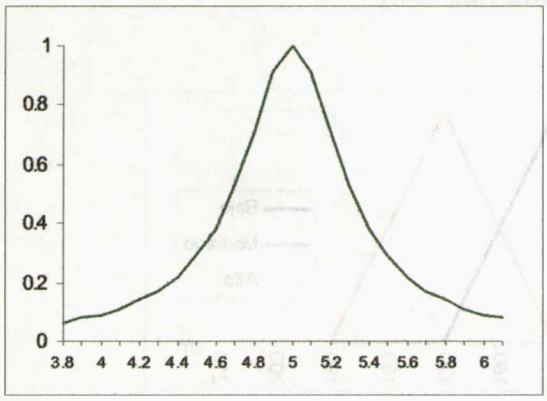
Sea ahora el conjunto difuso $\bar{A}$ constituido por "números reales cercanos a 5". Entonces tenemos

$$\bar{A} = \{ (x, \mu_{\bar{A}}(x)) \mid x \in U \},$$

donde la función de pertenencia que en nuestro ejemplo define la característica de "ser cercano a 5" es

$\mu_A(x) = 1/(1 + 10\ (x - 5)^2$

y graficamos como sigue:



En el gráfico vemos que, por ejemplo, el real 4.7 es cercano al 5 en 0.6.

Las primeras diferencias que se hacen evidentes entre Conjuntos Clásicos y Conjuntos Difusos son las siguientes:

- La función de pertenencia asociada a los conjuntos clásicos sólo puede tener dos valores: 1 ó 0, mientras que en los conjuntos difusos puede tener cualquier valor entre 0 y 1.
- Un elemento puede pertenecer (parcialmente) a un conjunto difuso y simultáneamente pertenecer (parcialmente) al complemento de dicho conjunto. Lo anterior no es posible en los conjuntos clásicos, ya que constituiría una violación al principio del tercero excluido.
- Las "fronteras" de un conjunto clásico son "terminantes", en tanto que las de un conjunto difuso son, precisamente, "difusas", ya que pueden existir elementos en las fronteras mismas.

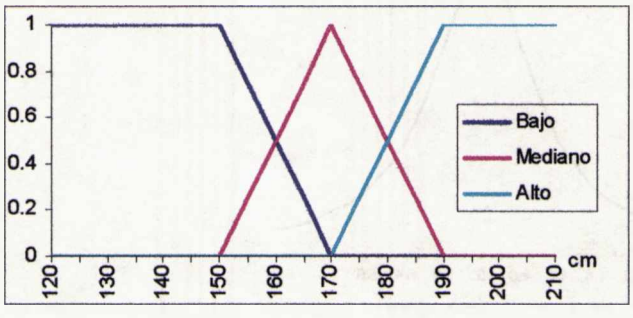
¿Qué sentido puede tener pertenecer "parcialmente" a un conjunto difuso? En muchos casos puede tener más sentido que pertenecer totalmente a un conjunto clásico.

Ejemplo:

Supóngase que se desea clasificar a los miembros de un equipo de basquet según su estatura en tres conjuntos: Bajos, Medianos y Altos. Podría plantearse que se es Bajo si se tiene una estatura inferior a, 160 cm, que se es mediano si la estatura es superior a 160 cm e inferior a 180 cm, y se es alto si la estatura es superior o igual a 180 cm, con lo que se lograría una clasificación en conjuntos clásicos. Esta es una clasificación taxativa de los jugadores.

Sin embargo, puede argumentarse que la diferencia que existe entre dos jugadores, uno con estatura de 179.9 cm y el otro de 180.0, cm para este contexto del ejemplo es insignificante. Ese milímetro de diferencia hace que los dos jugadores queden según la clasificación previa, rotulados con etiquetas distintas: uno es Mediano y el otro es Alto. Si se opta por efectuar la clasificación con lógica difusa estos cambios abruptos se evitan, debido a que las "fronteras" entre los conjuntos difusos permiten cambios graduales en la clasificación.

En la figura siguiente vemos como podría hacerse tal clasificación: El universo de discurso es el conjunto continuo de todas las posibles estaturas (el intervalo [120cm,210cm] por ejemplo). Las funciones de pertenencia de cada uno de los tres conjuntos difusos que proponemos son:



Otra manera de representar un conjunto difuso es a través del *conjunto soporte* de un conjunto difuso.

Definición: El conjunto soporte de un conjunto difuso A es el conjunto discreto de todos los $x \in U$ tal que $\mu_A(x) > 0$. Esto es,

$$Supp(A) = \{ x \in U \mid \mu_A(x) > 0\}.$$

Ejemplo:

Supongamos que para un examen las posibles calificaciones son U={10,20,30,40,50,60,70,80,90,100}. Consideremos tres posibles conjuntos difusos, A="Calificación alta", B="Calificación media", y C="Calificación Baja", cuyas funciones de pertenencia son definidas abajo:

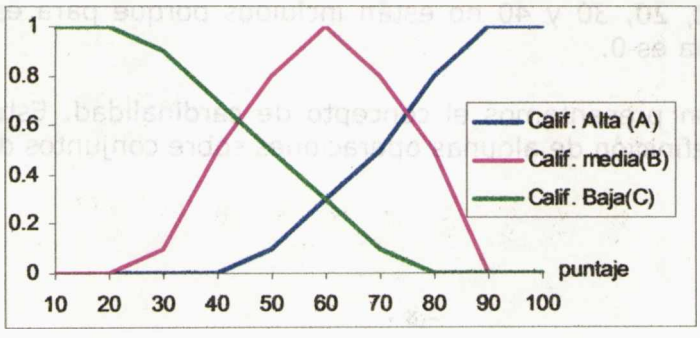
Calificación numérica	Calif. Alta(A)	Calif. media(B)	Calif. Baja(C)
10	0	0	1
20	0	0	1
30	0	0.1	0.9
40	0	0.5	0.7
50	0.1	0.8	0.5
60	0.3	1	0.3
70	0.5	0.8	0.1
80	0.8	0.5	0
90	1	0	0
100	1	0	0

La Tabla debe interpretarse como sigue: la calificación 10 es baja con grado de pertenencia 1, y el grado de pertenencia a alta y media es cero; la calificación 60 es entendida como media con grado de pertenencia 1, pero también es alta con grado de pertenencia 0.3 y baja con grado de pertenencia 0.3. Así para todas las calificaciones.

Entonces tenemos:

Supp(A) = Supp("Calificación alta") = {50,60,70,80,90,100}
Supp(B) = Supp("Calificación media") = {30,40,50,60,70,80}
Supp(C) = Supp("Calificación baja") = {10,20,30,40,50,60,70}

El gráfico correspondiente es:

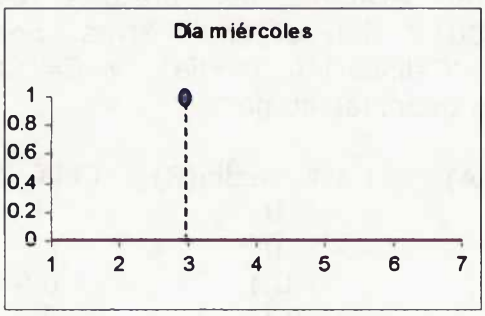


Definición: Un conjunto difuso A cuyo soporte es un punto simple en U con $\mu_A(x) = 1$ es denominado como un *singleton difuso* (este concepto será usado más adelante).

Ejemplo: representamos el día miércoles como un singleton difuso, donde el dominio son los días de la semana.

Supp("miércoles") = {3}

Gráficamente:



Para un universo discreto de discurso $U = \{x_1, x_2, \ldots, x_n\}$, un conjunto difuso puede ser representado usando el concepto de pares ordenados y escrito como

$$A = \{(x_1, \mu_A(x_1)), (x_2, \mu_A(x_2)), \ldots, (x_n, \mu_A(x_n))\}.$$

Usando el soporte de un conjunto difuso A, podemos simplificar la notación:

$$A = \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_i/x_i + \ldots + \mu_n/x_n = \sum \mu_i/x_i,$$

Donde + indica la unión de elementos y μ_i es el grado de pertenencia de x_i al conjunto, esto es, $\mu_i = \mu_A(x_i) > 0$. Por ejemplo, en el ejemplo anterior:

B = "Calificación media" = 0.1/30 + 0.5/40 + 0.8/50 + 1/60 + 0.8/70 + 0.5/80.

Notar que en el uso del soporte de un conjunto difuso para representar un conjunto difuso, consideramos solamente aquellos elementos en el universo de discurso que tienen un grado de pertenencia distinto de 0 en el conjunto difuso. Por ejemplo, vemos que en el ejemplo mostrado anteriormente de las calificaciones

Supp(A) = Supp("Calificación alta") = {50,60,70,80,90,100}

Notar que 10, 20, 30 y 40 no están incluidos porque para estos valores el grado de pertenencia es 0.

A continuación presentamos el concepto de cardinalidad. Este concepto es necesario para la definición de algunas operaciones sobre conjuntos difusos.

Definición: La cardinalidad de un conjunto difuso es la suma de los grados de pertenencia de todos los elementos de x en A:

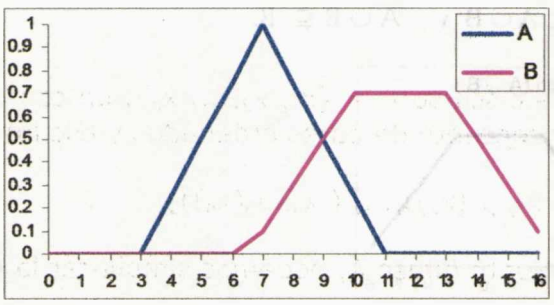
$$|A| = \sum_{x \in U} \mu_A(x)$$

Ejemplo:

Consideremos el conjunto difuso A (Calificación alta) del ejemplo de las calificaciones.

$$|A| = |\text{Calificación alta}| = 0.1 + 0.5 + 0.8 + 1 + 1 = 3.7$$

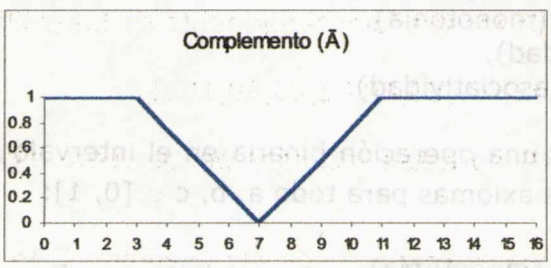
Introducimos seguidamente algunas definiciones básicas de teoría de conjuntos y operaciones sobre conjuntos difusos. Sea A y B dos conjuntos difusos en el universo de discurso U; graficados a continuación:



Utilizando como referencia los conjuntos difusos A y B, ilustramos las siguientes operaciones:

1. Complemento: Cuando $\mu_A(x) \in [0,1]$, el complemento de A, denotado como $\bar{A}$, es definido por su función de pertenencia como

$$\mu_{\bar{A}}(x) =_{\text{def}} 1 - \mu_A(x) \quad \forall x \in U.$$

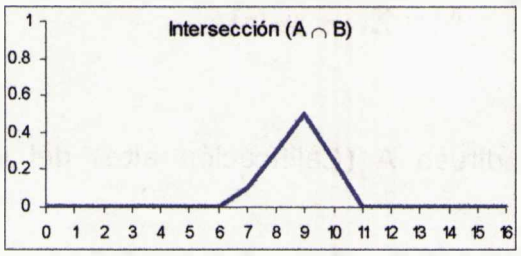


2. Intersección: La intersección de dos conjuntos difusos A y B, denotada $A \cap B$, es definida por

$$\mu_{A \cap B}(x) =_{\text{def}} \min [\mu_A(x), \mu_B(x)] \equiv \mu_A(x) \wedge \mu_B(x) \quad \forall x \in U,$$

donde $\wedge$ indica la operación mínimo. Es claro que

$$A \cap B \subseteq A \quad y \quad A \cap B \subseteq B$$

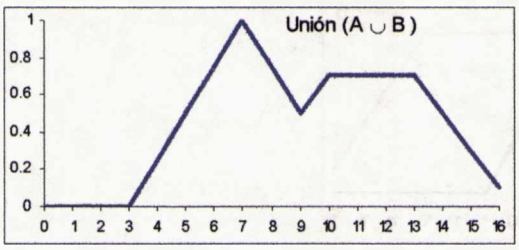


3. Unión: La unión de dos conjuntos difusos A y B, denotada $A \cup B$, es definida por

$$\mu_{A \cup B}(x) =_{def} \max [\mu_A(x), \mu_B(x)] \equiv \mu_A(x) \vee \mu_B(x) \quad \forall x \in U,$$

donde $\vee$ indica la operación máximo. Es claro que

$$A \subseteq A \cup B \text{ y } \quad A \cup B \subseteq B.$$



Las operaciones de unión e intersección sobre conjuntos difusos son frecuentemente llamadas "Normas triangulares" (t-normas) y "Conormas triangulares" (t-conormas).

Definición: Una t-norma T es una operación binaria en el intervalo unitario que satisface al menos los siguientes axiomas para todo a, b, c $\in$ [0, 1]:

T (a, 1) = a (elemento neutro),
b $\leq$ c implica T (a, b) $\leq$ T (a, c) (monotonía),
T (a, b) = T (b, a) (conmutatividad),
T (a, T (b, c)) = T (T (a, b), c) (asociatividad).

Definición: Una t-conorma S es una operación binaria en el intervalo unitario que satisface al menos los siguientes axiomas para todo a, b, c $\in$ [0, 1]:

S (a, 0) = a (elemento neutro),
b $\leq$ c implica S (a, b) $\leq$ S (a, c) (monotonía),
S (a, b) = S (b, a) (conmutatividad),
S (a, T (b, c)) = S (S (a, b), c) (asociatividad) . [BAB2001]

4. Igualdad: A y B son iguales si y solo si

$$\mu_A(x) = \mu_B(x) \quad \forall x \in U.$$

Si $\mu_A(x) \neq \mu_B(x)$ para algún $x \in U$, entonces $A \neq B$. Esta definición de igualdad es discreta. Para chequear el grado de igualdad de dos conjuntos difusos, usaremos la *medida de similitud* [Lin and Lee, 1992]:

$$E(A,B) \equiv \text{grado } (A = B) =_{def} \frac{|A \cap B|}{|A \cup B|},$$

donde $\cap$ y $\cup$ denotan intersección y unión de A y B, respectivamente. Cuando $A = B$, $E(A,B) = 1$; cuando $|A \cap B| = 0$, $E(A,B) = 0$. En el caso más general, $0 \leq E(A,B) \leq 1$.

5. Subconjunto: A es subconjunto de B; esto es, $A \subseteq B$ sí y solo sí

$$\mu_A(x) \leq \mu_B(x) \quad \forall x \in U.$$

Si $A \subseteq B$ y $A \neq B$, entonces A es un subconjunto propio de B; esto es, $A \subset B$. De nuevo, la definición de subconjunto es discreta. Para chequear el grado en que A es subconjunto de B, podemos usar el coeficiente [Kosko, 1992a]:

$$S(A,B) \equiv \text{grado } (A \subseteq B) =_{def} \frac{|A \cap B|}{|A|}$$

Ejemplo:

Consideremos las funciones de pertenencia de los conjuntos difusos A (Calificación alta) y B (Calificación media) del ejemplo de las calificaciones. Obviamente $A \neq B$, A no es subconjunto de B ni B es subconjunto de A.

$\bar{A} = 1/10 + 1/20 + 1/30 + 1/40 + 0.9/50 + 0.7/60 + 0.5/70 + 0.2/80$

$A \cap B = 0.1/50 + 0.3/60 + 0.5/70 + 0.5/80$

$A \cup B = 0.1/30 + 0.5/40 + 0.8/50 + 1/60 + 0.8/70 + 0.8/80 + 1/90 + 1/100$

$$E(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{0.1 + 0.3 + 0.5 + 0.5}{0.1 + 0.5 + 0.8 + 1 + 0.8 + 0.8 + 1 + 1} = \frac{1.4}{6} = 0.23$$

$$S(A,B) = \frac{|A \cap B|}{|A|} = \frac{0.1 + 0.3 + 0.5 + 0.5}{0.1 + 0.3 + 0.5 + 0.8 + 1 + 1}$$

6. Ley de doble negación (involución): el complemento del complemento de un conjunto difuso A es el mismo conjunto difuso A.

$$\bar{\bar{A}} = A.$$

7. Leyes de DeMorgan:

El complemento de la unión de dos conjuntos difusos es la intersección de los complementos de cada conjunto.

$$\overline{A \cup B} = \bar{A} \cap \bar{B}.$$

El complemento de la intersección de dos conjuntos difusos es la unión de los complementos de cada conjunto.

$$\overline{A \cap B} = \bar{A} \cup \bar{B}$$

Sin embargo, la ley del tercero excluido ($E \cup \bar{E} = U$) y la ley de contradicción ($E \cap \bar{E} = \emptyset$) del conjunto discreto E no son válidas en conjuntos difusos. Esto es, para el conjunto difuso A,

$$\bar{A} \cup A \neq U \quad \text{y} \quad \bar{A} \cap A \neq \emptyset,$$

lo cual significa que debido a la falta de límites precisos, los conjuntos complementarios se superponen (como vimos en el ejemplo de la altura de jugadores del equipo de básquet) y no pueden cubrir el conjunto universal U perfectamente. Por el contrario, estas leyes son las características necesarias para conjuntos clásicos.

2.1.3- El principio de extensión

Definición: Dada una función f: $U \rightarrow V$ y un conjunto difuso A en U, donde A $= \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_n/x_n$ el principio de extensión establece que:

$f(A) = f(\mu_1/x_1 + \mu_2/x_2 + ... + \mu_n/x_n) = \mu_1 / f(x_1) + \mu_2 / f(x_2) + ... + \mu_n / f(x_n)$

El principio de extensión, fue introducido por Zadeh[1978a]. Éste principio permite la generalización de conceptos matemáticos discretos en un marco de conjuntos difusos y extiende la aplicación punto a punto en una aplicación para conjuntos difusos. Éste proporciona un mecanismo por el cual cualquier función *f* que se aplica a una n-tupla (x_1, x_2, ..., x_n) en el conjunto discreto U a un punto en el conjunto discreto *V* para ser generalizado a una aplicación de n conjuntos difusos U a un subconjunto difuso en *V*. Por lo que cualquier relación matemática entre elementos no difusos puede ser extendida para ser tratada como entidades difusas.

Ejemplo:

Sea $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Un conjunto difuso A = "Grande" es definido como

A = "Grande" = 0.5/6 + 0.7/7 + 0.8/8 + 0.9/9 + 1/10.

Si la función *f* es una operación de potencia cuadrada y es indicada $y = f(x) = x^2$, entonces por el principio de extensión, el conjunto difuso B = "grande"^{2}, puede ser fácilmente calculado como

B = "grande"^{2} = 0.5/36 + 0.7/49 + 0.8/64 + 0.9/81 + 1/100.

2.1.4- Variables lingüísticas

Una variable lingüística toma valores que son palabras o sentencias en un lenguaje natural o artificial. Por ejemplo, velocidad es una variable lingüística, tomando valores tales como despacio, rápido, muy rápido.

El concepto de variable lingüística fue introducido por Zadeh [1975a] para proporcionar un medio de caracterización aproximada de fenómenos que pueden considerarse demasiado complejos o demasiado mal definidos para ser tratados con una descripción en términos cuantitativos convencionales.

Antes de introducir la definición formal de variable lingüística, definimos el concepto de variable difusa.

Definición: Una variable difusa es caracterizada por una tripla (X, U,R(X)) donde X es el nombre de la variable, U es el universo de discurso, y R(X) es un subconjunto difuso de U que representa una restricción difusa impuesta por X.

Ejemplo:

X = "Anciano",

U = {10, 20, 30, 40, 50, 60, 70, 80}, conjunto discreto de edades,

R(X) = 0.1/20 + 0.2/30 + 0.4/40 + 0.5/50 + 0.8/60 +1/70 + 1/80 es una restricción difusa sobre "Anciano" (Se observa que 10 no aparece en R(X) lo que significa que su valor de pertenencia es 0.0).

Una variable lingüística es una variable de más alto orden que una variable difusa: toma variables difusas como valores. Es caracterizada por una quíntupla (x, T(x), U, G, M) en la cual x es el nombre de la variable; T(x) es el conjunto de términos de x, esto es, el conjunto de nombres de valores lingüísticos de x con cada valor siendo una variable difusa sobre U; G es una regla sintáctica para generación de nombres de valores de x; y M es una regla semántica para asociar cada valor de x con su significado.

Ejemplo:

Sea "velocidad" una variable lingüística con U=[0,200], esto es, x="velocidad", entonces

T(velocidad)= { Despacio, Moderado, Ligero, ...}

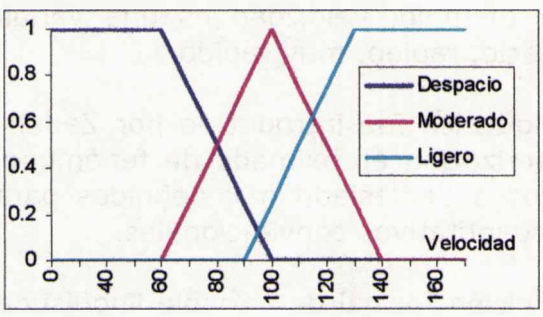
La regla sintáctica G para generar los nombres de los elementos en T(velocidad) es bastante intuitiva. La regla semántica M puede ser definida como:

M(Despacio)= el conjunto difuso para "velocidad por debajo de los 70 km por hora" con función de pertenencia $\mu_{Despacio}$.

M(Moderado)= el conjunto difuso para "velocidad cercana a los 100 km por hora" con función de pertenencia $\mu_{Moderado}$.

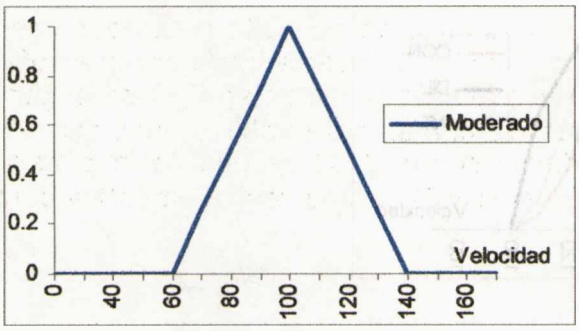
M(Ligero)= el conjunto difuso para "velocidad arriba de los 130 km por hora" con función de pertenencia μ_{Ligero}.

Gráficamente:



Vamos ahora a introducir el modificador lingüístico (o borde) h. Es un operador para modificar el significado de un conjunto difuso A para crear un nuevo conjunto difuso h(A). Por ejemplo en el conjunto difuso "MUY joven", "MUY" es un modificador lingüístico. Las siguientes operaciones de conjuntos difusos son frecuentemente usadas para definir modificadores lingüísticos:

Sea el conjunto difuso "Moderado"



1- Concentración: Su función de pertenencia tomara valores más pequeños, concentrándose en los valores mayores

CON(A)

$$\mu_{con(A)}(\mu) = (\mu A(\mu))^2.$$

2- Dilación: Efecto contrario a la concentración

DIL(A)

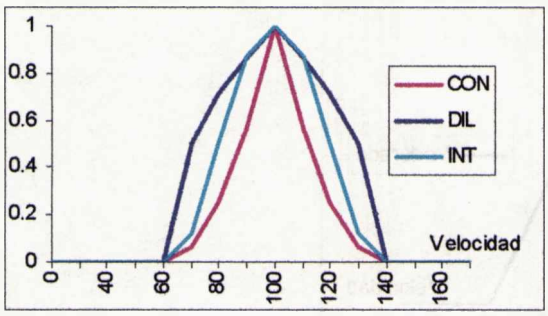
$$\mu_{dil(A)}(\mu) = (\mu A(\mu))^{1/2}.$$

3- Intensificación: Se disminuyen los valores menores a 0.5 y se aumentan los mayores

INT(A)

$$\mu_{dil(A)}(\mu) = \begin{cases} 2(\mu_A(\mu))^2. & \mu_A(\mu) \in [0, 0.5] \\ 1 - 2(1 - \mu A(\mu))^2. & \text{en otro caso.} \end{cases}$$

Vemos gráficamente el resultado de aplicar concentración, dilación e intensificación sobre el conjunto "Moderado":



Algunos populares modificadores lingüísticos son:

Arriba (ABOVE): identifica el valor x al cual el máximo valor es alcanzado. Todos los valores de pertenencia debajo de este punto tomarán el valor de cero y todos los valores de pertenencia por arriba de este valor tomarán el valor $1-\mu_A(x)$.

Abajo (BELOW): identifica el valor x al cual el máximo valor es alcanzado. Todos los valores de pertenencia arriba de este punto tomarán el valor de cero y todos los valores de pertenencia por abajo de este valor tomarán el valor $1-\mu_A(x)$.

Extremadamente(EXTREMELY): eleva el factor de pertenencia por un factor de 3: $(\mu_A(\mu))^3$

Mas o menos(MOREORLESS): eleva el factor de pertenencia por un factor de 1/3: $(\mu_A(\mu))^{1/3}$

No(NOT): obtiene el complemento del fuzzySet dado: $1-\mu_A(x)$

Más(PLUS): eleva el factor de pertenencia por un factor de 1.25: $(\mu_A(\mu))^{1.25}$.

Algo(SOMEWATH): eleva el factor de pertenencia por un factor de ½: $(\mu_A(\mu))^{1/2}$.

Muy(VERY): eleva el factor de pertenencia por un factor de 2: $(\mu_A(\mu))^2$.

Ligeramente (SLIGHTLY): Intensificación [*Más*(A) AND *NOT*(*Muy*(A)))]

2.1.5- Valores de verdad en lógica difusa

Hemos dicho que la lógica difusa es una extensión de la lógica booleana de la teoría de conjuntos en la cual los valores de verdad son términos de la variable lingüística "verdad". Su objetivo es proveer fundamentos para razonamiento aproximado con una proposición imprecisa usando teoría de conjuntos difusa.

Entonces, distinto a lógica booleana, a los valores de verdad en lógica difusa se les permite comprender los conjuntos difusos de intervalo de unidad [0,1] o bien un punto en el intervalo. Por ejemplo, un valor de verdad en lógica difusa, "MUY verdadero", puede ser interpretado como un conjunto difuso en [0,1].

Definición: El valor de verdad de la proposición "X es A" (o simplemente el valor de verdad de A) denotado v(A), es un punto en [0,1] (llamado el valor de verdad numérico) o un conjunto difuso en [0,1] (llamados los valores de verdad lingüísticos).

A continuación mostramos cómo obtener el valor de verdad de una proposición. Los conectivos lógicos usados para la construcción de fórmulas proposicionales son los usuales: negación (NOT), conjunción ($\wedge$), disjunción ($\vee$) e implicación ($\Rightarrow$).

Si v(A) y v(B) son los valores de verdad (numéricos) de las proposiciones A y B, respectivamente, entonces

$$v(\text{NOT A}) =_{def} 1 - v(A),$$

$$v(\text{A AND B}) =_{def} v(A) \wedge v(B) = \min\{ v(A), v(B)\},$$

$$v(\text{A OR B}) =_{def} v(A) \vee v(B) = \max\{ v(A), v(B)\},$$

$$v(A \Rightarrow B) =_{def} v(A) \Rightarrow v(B) = \max\{1 - v(A), \min (v(A), v(B))\},$$

La implicación está basada en la definición dada para ésta por Zadeh. Pero hay otras definiciones de implicación.

Si v(A) y v(B) son valores de verdad lingüísticos de proposiciones A y B, respectivamente, expresados como $v(A) = \alpha_1 /v_1 + \alpha_2 /v_2 +...+ \alpha_i /v_i+...+ \alpha_n /v_n$ y $v(B) = \beta_1 /w_1 + \beta_2 /w_2 +...+ \beta_i /w_i+...+ \beta_n /w_n$, donde $\alpha_i, \beta_i, v_i, w_i \in [0,1]$, entonces usando el principio de extensión tenemos

$$v(\text{NOT A}) \equiv \alpha_1 /(1 - v_1) + \alpha_2 /(1 - v2)+...+ \alpha_i /(1 - vi)+...+ \alpha_n /(1 - v_n)$$

$$v(\text{A AND B}) \equiv v(A) \wedge v(B) = \sum_{ij} \min (\alpha_i, \beta_i)/\min (v_i, w_i),$$

$$v(\text{A OR B}) \equiv v(A) \vee v(B) = \sum_{ij} \min (\alpha_i, \beta_i)/\max (v_i, w_i),$$

$$v(A \Rightarrow B) \equiv v(A) \Rightarrow v(B) = \sum_{ij} \min (\alpha_i, \beta_i)/\max\{1 - v_i, \min(v_i, w_i)\},$$

Ejemplo:

Sea los valores de verdad lingüísticos de dos proposiciones

P = 0.5/0.6 + 0.7/0.7 + 1/0.8 + 1/0.9 + 1/1

Q = 0.6/0.8 + 1/0.9 + 0.6/1

Sean v(P) = "MORE OR LESS True" y v(Q) = "ALMOST True"

v(NOT P) = v(NOT MORE OR LESS True)

= 0.5/0.4 + 0.7/0.3 + 1/0.2 + 1/0.1 + 1/0

v(P AND Q) = v(P) ∧ v(Q) = "MORE OR LESS True" ∧ "ALMOST True"

= 0.5/0.6 + 0.7/0.7 + 1/0.8 + 1/0.9 + 0.6/1

v(P OR Q) = v(P) V v(Q) = "MORE OR LESS True" V "ALMOST True"

= 0.6/0.8 + 1/0.9 + 1/1

2.1.6- Operaciones y estructura básica de sistemas de control de lógica difusa

La arquitectura típica de un sistema CLD (Controlador de Lógica Difusa) consta de cuatro componentes principales: un fusificador, una base de reglas difusas, una máquina de inferencia y un defusificador (Figura 1). En algunos sistemas la salida del defusificador es una acción de control sobre un medio. En otros sistemas la salida del defusificador es una "decisión", entonces se trata de un sistema de decisión de lógica difusa. El fusificador tiene la tarea de transformar medidas de datos discretos (ejemplo la velocidad es 10 millas por hora) en valores lingüísticos adecuados (es decir conjuntos difusos, por ejemplo, la velocidad es demasiado lenta). La base de reglas difusas almacena el conocimiento empírico de cómo opera el proceso, este conocimiento es proporcionado por los expertos en el dominio (ejemplo la regla "Si la temperatura del motor es extremadamente caliente entonces apagar el motor" construida a partir del conocimiento de los operarios). La máquina de inferencia es el corazón de un CLD, y tiene la capacidad de simular la toma de decisiones humanas ejecutando razonamiento aproximado para lograr la estrategia de control deseada. El defusificador es usado para producir una decisión no difusa (ejemplo: la persona es 0.8 alta para su edad) o una acción de control de una acción de control difusa inferida por una máquina de inferencia (ejemplo: aumentar la temperatura de una caldera).

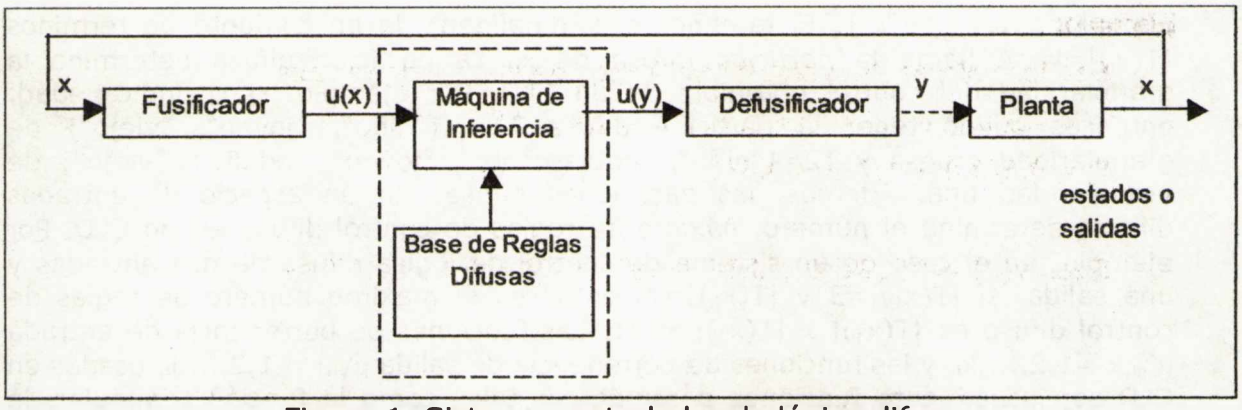


Figura 1. Sistema controlador de lógica difusa

2.1.7- Espacios de Entrada-Salida

El propósito de los controladores de lógica difusa es computar valores de variables de control (o acción) de los procesos controlados para obtener la performance deseada. Esto se realiza mediante la observación o medición de variables de estado. Así, es esencial una elección correcta de variables de estado y variables de control del proceso para la caracterización de un sistema de control de lógica difusa (SCLD). Esta elección tiene un efecto sustancial en la performance de un CLD. La experiencia de los expertos y los conocimientos de ingeniería juegan un importante rol en el proceso de selección de variables de control y variables de estado. Generalmente, las variables de entrada en CLD son el estado, el error de estado, el error de estado derivado, el error de estado integral. Siguiendo la definición de variables lingüísticas, el vector de entrada *x* el cual incluye las variables lingüísticas de estado de entrada x_i y el vector de estado de salida *y* el cual incluye las variables lingüísticas de estado (o control) de salida y_i, pueden ser definidos, respectivamente, como

$$X = \{(x_i, U_i, \{T^1_{xi}, T^2_{xi}, \ldots, T^{ki}_{xi}\}, \{\mu^1_{xi}, \mu^2_{xi}, \ldots, \mu^{ki}_{xi}\}) \mid_{i=1,\ldots,n}\}, \quad (1)$$

$$Y = \{(y_i, U_i, \{T^1_{yi}, T^2_{yi}, \ldots, T^{ki}_{yi}\}, \{\mu^1_{yi}, \mu^2_{yi}, \ldots, \mu^{ki}_{yi}\}) \mid_{i=1,\ldots,m}\}, \quad (2)$$

Donde las variables lingüísticas de entrada x_i constituyen un espacio de entrada $U = U_1 \times U_2 \times \ldots \times U_n$ y las variables lingüísticas de salida y_i constituyen un espacio de salida $V = V_1 \times V_2 \times \ldots \times V_m$. De (1) y (2), se observa que una variable lingüística de entrada x_i en un universo de discurso U_i está caracterizada por $T(x_i)=\{T^1_{xi}, T^2_{xi}, \ldots, T^{ki}_{xi}\}$ y $\mu(x_i)=\{\mu^1_{xi}, \mu^2_{xi}, \ldots, \mu^{ki}_{xi}\}$, donde $T(x_i)$ es el conjunto de términos de x_i, esto es, el conjunto de nombres de valores lingüísticos de x_i, con cada valor T^{ki}_{xi} siendo un número difuso con función de pertenencia μ^{ki}_{yi} definido sobre U_i. Así $\mu(x_i)$ es una regla semántica para asociar cada valor con su significado. Por ejemplo, si x_i indica velocidad, entonces $T(x_i)=\{T^1_{xi}, T^2_{xi}, T^3_{xi}\}$ puede ser "lento", "medio" y "rápido". Similarmente, una variable lingüística de salida y_i es asociada con un conjunto de términos $T(y_i)=\{T^1_{yi}, T^2_{yi}, \ldots, T^{ki}_{yi}\}$ y

$\mu(y_i)=\{\mu^1_{yi}, \mu^2_{yi},..., \mu^{ki}_{yi}\}$. El tamaño (o cardinalidad) de un conjunto de términos $|T(x_i)|=k_i$ se llama la *partición difusa* de x_i. La partición difusa determina la granularidad del control obtenible de un CLD. Por ejemplo, si x_i indica edad, entonces puedo tener la partición difusa T1= {"niño", "joven", "viejo"} de granularidad gruesa y T2={"niño", "adolescente", "joven", "adulto", "viejo"} de granularidad fina. Además, las particiones difusas en un espacio de entradas difusas determina el número *máximo* de reglas de control difuso en un CLD. Por ejemplo, en el caso de un sistema de control de lógica difusa de dos entradas y una salida, si $|T(x_1)|=3$ y $|T(x_2)|=7$, entonces el máximo número de reglas de control difuso es $|T(x_1)| \times |T(x_2)| = 21$. Las funciones de pertenencia de entrada $\mu^k_{xi}, k = 1,2,...,k_i$, y las funciones de pertenencia de salida $\mu^l_{yi}, l = 1,2,...,l_i$, usadas en CLD son usualmente funciones paramétricas tales como la función triangular, la función trapezoidal, etc. Una partición difusa adecuada de los espacios de entrada y salida y una elección correcta de las funciones de pertenencia juegan un rol esencial para alcanzar un exitoso diseño de CLD. Desafortunadamente, ellos son no determinísticos y no tienen una solución única. Tradicionalmente, un procedimiento de prueba y error heurístico es usado para determinar una partición difusa óptima. Además, la elección de las funciones de pertenencia está basada en un criterio de decisión subjetivo y consume un alto tiempo de prueba y error [NFS 1996].

2.1.8- Fusificador

Un fusificador realiza la función de fusificación. Ésta no es más que una valuación para transformar los datos medidos. Puede ser definido como un mapeo del espacio de entrada observado a términos de un conjunto difuso en un universo de discurso de entrada especificado. La fusificación es necesaria y deseable en un estado temprano. En aplicaciones de control difuso, los datos observados son usualmente discretos. Una aproximación simple y natural de fusificación es convertir un valor discreto x_0 en un singleton difuso A dentro del universo de discurso especificado A. Esto es, la función de pertenencia A, $\mu_A(x)$, es igual a 1 en el punto x_0, y cero en otros lugares. En este caso, para un valor específico $x_i(t)$ al momento t, este es mapeado al conjunto difuso T^1_{xi} con grado $\mu^1_{xi}(x_i(t))$ y al conjunto difuso T^2_{xi} con grado $\mu^2_{xi}(x_i(t))$, y así siguiendo. Este enfoque es ampliamente usado en CLD porque simplifica enormemente el proceso de razonamiento difuso.

2.1.9- Base de Reglas Difusas

Las reglas de control difuso son una colección de reglas difusas IF-THEN en las cuales las condiciones y los consecuentes involucran variables lingüísticas. Esta colección de reglas de control difuso (o declaraciones de control difuso) caracteriza la relación de los sistemas de entrada-salida simple. La forma general de las reglas de control difuso en el caso de sistemas de entrada múltiple y salida simple es:

R_i = IF x is A_i ,..., AND y is B_i Then z = C_i, i =1,2,...,n, (4)

Donde x,...,y, y z son variables lingüísticas representando las variables de estado del proceso y las variables de control del proceso, respectivamente, y A_l,...,B_l, y C_l son los valores lingüísticos de las variables lingüísticas x,...,y, y z en el universo de discurso U,...,V, y W, respectivamente. Una variante de este tipo es que el consecuente sea representado como una función de las variables de estado del proceso x,...,y, esto es

R_l = IF x is A_l ,..., AND y is B_l Then z = $f_l(x,...,y)$, (5)

Donde $f_l(x,...,y)$, es una función de las variables de estado del proceso x,...,y. Las reglas de control difuso en las ecuaciones (4) y (5) evalúan el estado del proceso (es decir, estado, error de estado, error integral de estado, y así siguiendo) al momento t y computan y deciden las acciones de control como una función de las variables de estado (x,...,y). Vale la pena indicar que ambas reglas de control difuso tienen valores lingüísticos como entrada y valores lingüísticos [como en (4)] o valores discretos [como en (5)] como salida.

2.1.10- Máquina de inferencia

Este es el corazón del CLD en el modelado de la toma de decisiones "humanas" dentro del marco conceptual de la lógica difusa y razonamiento aproximado. El Modus Ponens Generalizado juega un rol especialmente importante en este contexto. Puede ser escrito como sigue:

Premisa 1: IF x is A, THEN y is B (6)

Premisa 2: x is A'

Conclusión: y is B'.

donde A, A', B, y B' son predicados difusos (conjuntos difusos o relaciones) en los conjuntos universales U, U, V, y V, respectivamente. En general, una regla de control difuso es una relación difusa expresada como una implicación $R \equiv A \Rightarrow B$. De acuerdo a la regla inferencia composicional la conclusión B' puede ser obtenida tomando la composición del conjunto difuso A' y la relación (aquí la relación es una relación difusa) $A \Rightarrow B$:

$$B' = A' \circ R = A' \circ (A \Rightarrow B). \quad (7)$$

la regla inferencia de composicional mencionada dice:

$$\frac{\begin{array}{c}X \text{ es } A \\ (X,Y) \text{ es } R\end{array}}{Y \text{ es } A \circ R}$$

donde $A \circ R$ indica la composición de un conjunto difuso A y una regla difusa R.

Hay cuatro tipos de operadores composicionales que pueden ser usados en la regla de inferencia composicional. Estos corresponden a las cuatro operaciones asociadas con las t-normas:

- Operación max-min [Zadeh, 1973]
- Operación producto máximo [Kaufmann, 1975]
- Operación producto limitado máximo (max – $\odot$) [Mizumoto, 1981]
- Operación producto drástico máximo (max – $\wedge$) [Mizumoto, 1981]

En aplicaciones de CLD, max-min y operadores composicionales de producto máximo son los más comúnmente usados debido a su simplicidad computacional y eficiencia. Sea max - $\star$ representando cualquiera de los cuatro operadores de composición. Entonces la ecuación (7) llega a ser

$$B' = A' \star R = A' \star (A \Rightarrow B). \qquad (8)$$

$$\mu_{B'}(v) = \sup_u \{\mu_A(u) \star \mu_{A \Rightarrow B}(u,v)\},$$

donde $\star$ denota las operaciones de t-norma tal como min, producto, producto limitado y producto drástico. Para la implicación $A \Rightarrow B$, hay cerca de 40 funciones de implicación difusa distintas descriptas en la literatura existente. En la tabla siguiente se muestran algunas de las más comúnmente usadas.

Regla de implicación difusa	Fórmulas de implicación	Implicación difusa
R_c: operación mínimo [Mamdani]	$a \Rightarrow b = a \wedge b$	$= \mu_A(u) \wedge \mu_B(v)$
R_p: operación producto [Larsen]	$a \Rightarrow b = a \cdot b$	$= \mu_A(u) \cdot \mu_B(v)$
R_{bp}: producto limitado	$a \Rightarrow b = 0 \vee (a + b - 1)$	$= 0 \vee [\mu_A(u) + \mu_B(v) - 1]$
R_{dp}: producto drástico	$a \Rightarrow b = \begin{cases} a & b=1 \\ b & a=1 \\ 0 & a,b<1 \end{cases}$	$= \begin{cases} \mu_A(u) & \mu_B(v)=1 \\ \mu_B(v) & \mu_A(u)=1 \\ 0 & \mu_A(u)\mu_B(v)<1 \end{cases}$
R_m:regla Lukasiewicz	$a \Rightarrow b = 1 \wedge (1 - a + b)$	$=1 \wedge (1- \mu_A(u) + \mu_B(v))$
R_m:regla max-min [Zadeh]	$a \Rightarrow b = (a \wedge b) \vee (1 - a)$	$= (\mu_A(u) \wedge \mu_B(v)) \vee (1- \mu_A(u))$

Las reglas de implicación difusa definidas de la tabla son generadas de la conjunción difusa ($\mu_{A\cap B}(x) = \min [\mu_A(x), \mu_B(x)] = \mu_A(x) \wedge \mu_B(x) \quad \forall x \in U$), disyunción difusa ($\mu_{A\cup B}(x) = \max [\mu_A(x), \mu_B(x)] = \mu_A(x) \vee \mu_B(x) \quad \forall x \in U$) o implicación difusa usando varias t-normas o t-conormas.

Resultados de estudios dicen que si bien las reglas Rc y Rp no tienen una estructura lógica bien definida, son las más adecuadas para modelar el Modus Ponens generalizado. Rm no es la más adecuada para razonamiento aproximado ya que el consecuente inferido no es el que más se adapta a nuestra intuición [NFS1996].

2.1.11- Defusificador

La defusificación es un mapeo desde un espacio de acciones de control difuso definidas sobre un universo de discurso de salida en un espacio de acciones de control no difuso (discreto). Este proceso es necesario debido a que en muchas aplicaciones prácticas son requeridas acciones de control discreto para activar el control.

Una estrategia de defusificación tiene como objetivo producir una acción de control difuso que mejor represente la distribución de posibilidad de una acción de control difuso inferida. Desafortunadamente, no hay un procedimiento sistemático para elegir una estrategia de defusificación. Dos métodos de defusificación usados comúnmente son el centro de área (CDA) y el método medio de máximo (MDM).

La estrategia de defusificación llamada "centro de área" genera el centro de gravedad de la distribución de posibilidad de una acción de control. En el caso de un universo discreto, este método da

$$Z^*_{CDA} = \frac{\sum_{j=1}^{n} \mu_C(z_j)\, z_j}{\sum_{j=1}^{n} \mu_C(z_j)}$$

donde n es el número de niveles de cuantificación de la salida, z_j es la cantidad de salidas de control en el nivel de cuantificación j, y $\mu_C(z_j)$ representa el valor de pertenencia en el conjunto difuso de salida C. Si el universo de salida es continuo, entonces la estrategia CDA genera una acción de control de salida de

$$Z^*_{CDA} = \frac{\int_z \mu_C(z) z \, dz}{\int_z \mu_C(z) \, d}$$

La estrategia MDM genera una acción de control que representa el valor medio de todas las acciones de control locales cuya función de pertenencia alcanza el máximo. En el caso de un universo discreto, la acción de control puede ser expresado como

$$Z^*_{MDM} = \frac{\sum_{j=1}^{m} z_j}{m}$$

donde z_j es el valor soporte al cual la función de pertenencia alcanza el máximo valor $\mu_C(z_j)$ y m es el número de tales valores de soporte.

De estas dos estrategias de fusificación comúnmente usadas, la estrategia CDA ha dado mejores resultados [Braae and Rutherford, 1978].

2.1.12- Sistema de decisión de lógica difusa

Seguidamente mostramos la arquitectura de un "sistema de decisión de lógica difusa" (caso particular de un "sistema de controlador de lógica difusa" mostrado en la Figura 1).

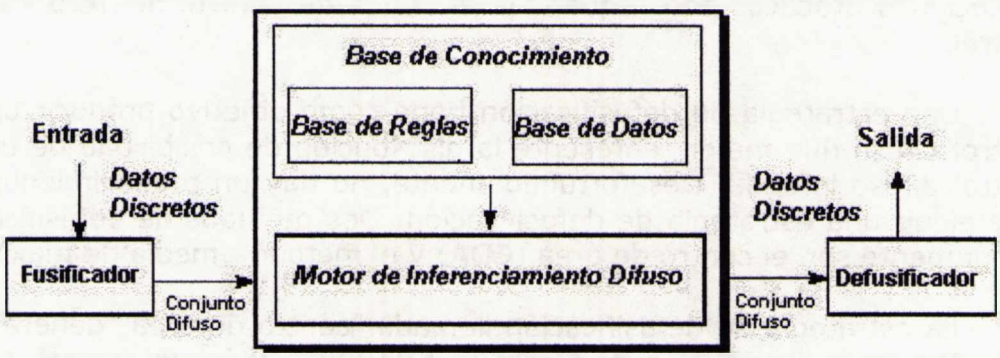


Figura 2 – Sistema de decisión de lógica difusa

Se realiza entonces un proceso en tres etapas:

Etapa de Fusificación

Este es el mecanismo de conversión de una entrada discreta a valores difusos necesarios para la evaluación de las reglas.

Etapa de Inferenciamiento

Inferencia: Una vez que las entradas han sido fusificadas, conocemos el grado en el cual cada parte del antecedente ha sido satisfecho para cada regla,

luego se computa el valor de verdad de las premisas de cada regla y se aplica a la parte de conclusión de la regla. Esto da como resultado un subconjunto difuso que será asignado a la variable de salida de cada regla.

Composición: Bajo composición todos los subconjuntos difusos asignados a cada variable de salida son combinados juntos para formar un subconjunto difuso simple para cada variable de salida.

Etapa de Defusificación

La entrada para el proceso de defusificación es un conjunto difuso (la salida de la etapa de agregación) y la salida es un número discreto.

El componente "Base de Reglas" (Figura 2) almacena el conocimiento empírico de los expertos en el dominio en cuestión.

2.2- Marketing y Publicidad en la Web

La palabra marketing (comercialización) proviene del vocablo inglés market (mercado) que representa un grupo de vendedores y compradores deseosos de intercambiar bienes y/o servicios. Por supuesto, suele ser necesaria cierta negociación. Esto se puede hacer personalmente por los interesados, en algún lugar físico o, de manera indirecta, por medio de una complicada red de intermediarios que vincula a los vendedores y compradores separados por grandes distancias [FC].

El marketing es el arte de encontrar, desarrollar y aprovechar las oportunidades. Definimos una oportunidad de marketing como un aspecto de la necesidad e interés del comprador en el cual hay una alta probabilidad de que una compañía pueda sacar provecho satisfaciendo esa necesidad[Kot 1999].

Una estrategia comercial especifica un mercado meta y una mezcla comercial afín. Es un cuadro de lo que la firma hará en cierto mercado. Se necesitan dos partes relacionadas entre ellas:

1- Un mercado meta, esto es, un grupo bastante homogéneo (similar) de clientes a los que la firma desea atraer.
2- Una mezcla comercial, esto es, las variables controlables que la empresa concierta para satisfacer este mercado meta.

Existen muchas formas de satisfacer las necesidades de los clientes meta. Es útil reducir todas las variables de la mezcla comercial a cuatro grupos básicos:

- Producto
- Promoción y publicidad
- Plaza
- Precio

Se crea un Producto para satisfacer a los clientes metas. Se halla una forma de llegar hasta la Plaza de los clientes metas. Se utiliza la Promoción para explicarles a éstos (y a los intermediarios) sobre el producto que se ha diseñado para ellos. Y se fija un Precio luego de estimar la reacción posible del cliente ante la oferta y el costo total de hacérsela llegar a él [FC].

La promoción es comunicar información entre el vendedor y el comprador en cierne u otros integrantes del canal para influir en las actitudes y el comportamiento. La principal tarea de promoción del gerente comercial consiste en decir a los clientes que encontrarán el Producto adecuado en la Plaza precisa al precio justo [FC].

'Qué' comunica el gerente comercial está determinado por las necesidades y las actitudes de los clientes metas. 'Cómo' se entregan los mensajes depende de qué mezcla de los diversos métodos de promoción escoge el gerente comercial.

Existen varios métodos de promoción:

La venta personal entraña comunicación oral directa entre vendedores y clientes potenciales. Este método no es aplicable en nuestro trabajo debido a que no contamos con este tipo de comunicación.

La venta masiva es comunicarse con gran cantidad de clientes potenciales al mismo tiempo. Las dos formas de encarar una venta masiva son publicidad y difusión. 'Publicidad' es toda forma pagada de presentación impersonal de ideas, bienes o servicios por un patrocinador identificado. 'Difusión' es cuando esto se realiza en forma gratuita. Nuestro trabajo esta orientado básicamente a la 'Publicidad' ya que los auspiciantes contratan espacios de publicidad que tienen un costo asociado.

La promoción de ventas se refiere a actividades de promoción (aparte de publicidad, difusión y venta personal) que estimulen el interés, la prueba del producto o la compra por el cliente final u otros miembros del canal. Es la promoción de ventas lo que influye sobre la conducta. El consumidor se entera de una venta, un ofrecimiento de dos artículos al precio de uno, un obsequio o una posibilidad de ganar algo. El anunciante puede realizar estas tareas de promoción cambiando temporalmente las reglas e incorporando distintos banners. Por ejemplo las reglas se pueden hacer más flexibles para que se muestren a más clientes.

Una propaganda es el acto de propagar o expandir algo, alguna cosa o alguna idea. Cuando esa propagación es de fin comercial, la identificamos como publicidad.

Un mensaje publicitario debe prestar atención a la sigla AIDA:

- llamar la Atención
- despertar Interés
- generar Deseo
- estimular a la Acción

Si un mensaje consigue llevar al navegante a esos estados mentales, una concreción de la venta esta garantizada. Sobre una óptica de Marketing, agregaríamos una letra más para que podamos "mantener" ese cliente que concreta una compra: la S de Satisfacción.

Esto es un gran desafío. Un consumidor, usuario o navegante presenta numerosos comportamientos. Es por esto que el marketing crea una estructura de agrupación, por criterio, de las personas. Aquí es donde surge el concepto de Segmentación.

Segmentar el mercado significa dividir en grupos menores (sub-mercados), identificarlos por criterios, con el fin de tornarlos homogéneos y más fácilmente manejables.

El marketing directo personalizado trata esencialmente de hacer llegar al público objetivo mensajes perfectamente adaptados a sus expectativas, bien sea información, ofertas o novedades de productos o servicios. Este tipo de acciones, cuando se realizan correctamente al target indicado con el mensaje apropiado son altamente eficaces. Para poder llevarlas a cabo se debe implantar un sistema de marketing "uno a uno", adaptar la oferta a la demanda de nuestros grupos de clientes. Esto implica inversión en tecnología, un dominio preciso en la gestión de perfiles de usuarios, creatividad y flexibilidad para mantener un alto dinamismo en diseño y selección de la oferta de productos y servicios.

Nuestro sistema pretende orientarse al marketing ya que intenta ofrecer:

- los productos más adecuados: cada sitio contará con cierta cantidad de anunciantes de productos entre los cuales se seleccionarán los que tengan mayor grado de adecuidad con el visitante.

- en los mercados idóneos: se seleccionan los clientes que tienen un alto grado de pertenencia al mercado meta del producto.

- mediante las promociones precisas: una vez seleccionados los anunciantes se pueden personalizar las publicidades si muestra la versión de estas que más se adecue a las características del visitante: ej. idioma, tecnología de la máquina cliente. Otra posibilidad es modificar temporalmente las reglas asociadas a los productos/servicios para impulsar las ventas por un determinado periodo de tiempo.

- dirigidas a personas específicas: los visitantes del sitio web en cuestión.

Para lograrlo utiliza un conjunto de reglas difusas elaborado por el anunciante e información personal de los visitantes del sitio.

Nuestro sistema muestra las publicidades en la página del sitio en cuestión mediante banners. El banner es la forma básica que toma la publicidad en Internet. Es un rectángulo que abarca una parte importante del ancho de la pantalla con un

mensaje promocional de alguna empresa, producto, lanzamiento, etc. Tienen como primer objetivo incrementar la cantidad de visitas al web site de la empresa o producto. Se busca que quien vea el banner ingrese a ese sitio. Un segundo objetivo es el mismo de la publicidad tradicional: lograr posicionamiento y recordación de la marca.

2.3- Agentes

El estudio de los agentes es un área de investigación en pleno y rápido desarrollo. El abuso producido en el uso del término "agente" ha llevado a enmascarar lo que en realidad es un campo amplio de investigación.

A pesar de que todavía no existe consenso para definir el concepto de agente, se han hecho ya importantes aproximaciones a la definición, diseño y construcción de diversos tipos de agentes.

Un agente es una entidad de software con un propósito específico. Una de las primeras definiciones aparecidas es: "un agente es algo que puede ser visto percibiendo su entorno y actuando sobre él [SR95]. Esta definición depende de lo que se considere como "entorno" y también de los que se considere como "percibir" y "actuar".

Un agente autónomo es un sistema que habita en un entorno dinámico y complejo, en el que percibe y actúa de manera relativamente independiente, alcanzando el conjunto de objetivos para los cuales fue diseñado [Mae95b]. Esta definición añade a la definición de agentes un aspecto importante como lo es la autonomía en el accionar y la capacidad de poseer un conjunto de objetivos.

La autonomía referida puede considerarse como la capacidad de que un agente tenga una actuación periódica, una ejecución espontánea e iniciativa, en la que el agente debe ser capaz de realizar acciones independientes o apropiativas que beneficiarán al usuario [Fon93].

Hay otras definiciones de agentes que tienen en cuenta el proceso de selección de acciones a realizar. Los agentes llamados "inteligentes" realizan tres acciones continuamente: perciben las condiciones de su entorno, actúan con el objetivo de modificarlo o sacar el mejor provecho de él y razonan para interpretar las percepciones, resolver problemas y determinar las acciones a realizar [HR95]. Otra manera de definirlo es llamarlo: "programa autocontenido que es capaz de controlar sus acciones y decisiones para alcanzar unos objetivos, basándose en su percepción del entorno" [NJ96]. O también: "Entidades de software que llevan a término un conjunto de operaciones bajo las órdenes de un usuario o de otro programa, con un cierto grado de independencia o autonomía, empleando conocimiento o representaciones de los objetivos del usuario [Gil95]".

Nuestro sistema es un agente inteligente en el sentido de [NJ96].

Capítulo 3

Nuestra propuesta: Selector de publicidad para la Web.

3.1- Arquitectura

Presentaremos una definición general de la arquitectura del selector. Son bien conocidos los beneficios de la independencia entre capas, escalabilidad y performance, por ello hemos optado por un modelo multi-tier, [CSGS].

Distribuimos los componentes de la aplicación en cuatro capas, a saber:

Donde:

- **Cliente:** está compuesta por el Web Browser que provee la interfaz con el usuario. Ésta contiene las páginas y el programa de configuración que brinda el servidor de Web.

- **Servidor de Web:** encargada de la interacción con el perfil de cliente usuario común tanto para resolver la selección de las publicidades como para configurar los datos personales de aquél.

- **Aplicación:** constituye la capa lógica del sistema. En ella se hallan todos los agentes encargados del proceso de selección de los anuncios y la aplicación de configuración.

- **Sistema de Gestión de Base de Datos:** se encarga de proveer los servicios para el acceso y mantenimiento de la información que genera la capa Aplicación. Responde a las peticiones de ésta.

A continuación se detallan las cuatro capas:

3.1.1- Cliente

3.1.1.1- Perfiles de cliente

El sistema permite la interacción con perfiles de clientes diferenciados por las operaciones que realizan en el browser de web.

a. Usuario común de Web: son los visitantes del sitio. Éstos interactúan con el servidor de Web navegando en las diferentes páginas del sitio Web o estableciendo los datos que lo caracterizan.

b. Usuario Administrador: constituido por los anunciantes. Éstos interactúan con el servidor de web realizando requerimientos de configuración de la aplicación. Un usuario con perfil Usuario Administrador establece parámetros que influyen en la selección de que anuncios se muestran para cada visitante del sitio en particular.

3.1.2- Componentes

Es donde se alojan las páginas que permiten la interacción del usuario con la aplicación, ya que recepciona las peticiones de la capa Cliente.

3.1.2.1- Componente común de Web:

Es el componente al cual acceden los clientes usuarios comunes (visitantes) al realizar la navegación por el sitio y su función es seleccionar las publicidades acordes con el perfil de éstos.

3.1.2.2- Componente administrador:

- Componente administrador de Web: es el componente mediante el cual los usuarios comunes "brindan conocimiento" al sistema sobre sí mismos (datos personales, demográficos, de estilo de vida, de sus compras, etc).

- Componente administrador de configuración de Web: es el componente mediante el cual los usuarios administradores (auspiciantes) establecen los datos necesarios para realizar la selección de anuncios.

3.1.3- Aplicación

Engloba la funcionalidad del sistema. Contiene la lógica necesaria para la interacción entre el usuario y el sistema. Está compuesta por dos elementos que se ejecutan en el browser web:

3.1.3.1- Aplicación de configuración: con la cual interactúa cada anunciante comportándose como Usuario Administrador. Ésta aplicación permite que el anunciante brinde al sistema las reglas difusas asociadas al producto que desea publicitar y los archivos de los banners que se mostrarán en los espacios destinados a publicidad en la página del sitio.

3.1.3.2- Aplicación de selección de anuncios: Los componentes de ésta se basan en el concepto de agente, quienes se relacionan con la capa superior (Servidor de Web), con la capa inferior (SGDB) y con otros agentes.

a. Agente Coordinador: es el único enlace entre la aplicación local y la capa superior. Se genera una instancia de éste automáticamente ante el ingreso del usuario al sistema y su objetivo es seleccionar anuncios y mostrarlos en el sitio que visita el usuario. Para esto utiliza la información brindada por los agentes auspiciantes.

b. Agentes Auspiciantes: Su objetivo es obtener un valor que determine el grado en que el producto / servicio auspiciado por el agente se adecua al visitante del sitio. Para esto combinan los resultados obtenidos de la ejecución de las reglas difusas definidas para ese producto / servicio.

c. Agentes Reglas Difusas: Representan a un visitante prototípico junto con la conclusión asociada a ese visitante. Cada regla se evalúa con los datos de la persona que en ese momento visita el sitio y como resultado obtiene un conjunto difuso que representa la salida de la regla.

3.1.4- Sistema de gestión de base de datos

Este componente conforma la capa inferior de la arquitectura y es la que posee la base del conocimiento. El SGBD resuelve las peticiones provenientes de la capa de aplicación.

El SGBD almacena información sobre usuarios (visitantes), productos/servicios, datos concernientes a la lógica difusa (reglas, conjuntos difusos, variables difusas). Cuenta con un proceso independiente que se ejecuta periódicamente cuya función es calcular el usuario promedio a partir de las características de los usuarios (edad, nacionalidad, ingresos, etc.) que ya están registrados en el sitio. El usuario promedio, para cada una de las características descriptivas, se obtiene de la siguiente forma:

- en el caso de tratarse de un valor numérico se realiza el promedio.
- en el caso de tratarse de un valor no numérico se toma el que aparece con más frecuencia.

Los motores de base de datos relacionales son muy escalables. En particular, para la implementación del selector, elegimos SQL Server 2000 Enterprise ya que puede admitir grupos de servidores de base de datos que cooperen para formar bases de datos con tamaño de terabytes a las que tienen acceso miles de usuarios al mismo tiempo, que podría ser de gran utilidad para el caso de sitios muy concurridos. Por lo tanto permite controlar el tráfico de cualquier sitio Web del mundo. Otra ventaja que nos proporciona este motor de base de datos es que se ajusta también a sí mismo, adquiriendo de manera dinámica recursos conforme se conectan más usuarios a la base de datos y liberando los recursos conforme se desconectan los usuarios.

3.2- Interfaz y funcionalidad

En las secciones siguientes describimos cuatro páginas web presentes en el sitio. Dos de ellas componen la interfaz del usuario común. Las dos restantes son accesibles únicamente al usuario administrador (anunciantes).

- Usuario común:
  - Página del sitio en cuestión: no existe interfaz gráfica ya que la funcionalidad de nuestro sistema se acopla al sitio al cual le brindamos nuestros servicios. Hemos logrado que las publicidades seleccionadas se muestren en las secciones definidas en la interfaz del sitio, adaptándose a todas las posibles distribuciones y tamaños de los espacios destinados para este fin.
  - Página de incorporación de datos del usuario común: Permitimos realizar tres tipos de operaciones:
    - Alta de usuario: se ingresan los datos personales y se asigna una clave para futuras modificaciones de estos.
    - Modificación de datos del usuario: Se ingresa a través de la clave brindada en el alta. Permite

realizar cambios sobre los datos definidos anteriormente.

- Baja de usuario: Se ingresa a través de la clave brindada en el alta. Permite la eliminación de usuarios existentes.

- Usuario administrador:

  - Página de parametrización: el objetivo de la misma es que los anunciantes gestionen los elementos que influyen en los cálculos de ranqueo de los anuncios disponibles e incorporen los banners que se mostrarán en la página del sitio. De aquí la importancia de la restricción de acceso.

    Los elementos que influyen en los cálculos de ranqueo son:

    - las reglas difusas: para definir las reglas puede hacer uso de las variables y conjuntos difusos predefinidos en el sitio que va a publicitar o crear sus propias definiciones de los mismos. El selector provee una amplia gama de modificadores difusos que pueden ser usados en la construcción de las reglas.

    - la cantidad de impresiones contratadas: se contrata un numero fijo de veces que se mostrará el banner.

    - el mínimo grado de adecuidad: establece cual es el mínimo grado en que la publicidad debe adecuarse al visitante.

    - cantidad máxima de anuncios por visitante: indica la cantidad máxima de veces que el anuncio será mostrado a cada visitante.

    - las secciones de la página donde acepta que se muestre su publicidad

    En el caso de los banners el anunciante deberá proveer los archivos de los mismos.

  - Página de consulta de datos estadísticos: permite que los auspiciantes consulten on-line la repercusión de la campaña publicitaria. Esta información se extrae de la base de datos y ésta se actualiza con cada acceso de los visitantes del sitio. Puede presentarse en formato gráfico o numérico. Algunos de

los reportes podrían ser: Cantidad de veces en que se mostró la publicidad, Cantidad de veces que despertó interés, momento del día en que se concentran la mayor cantidad de visitas.

Tener información acerca del visitante es de suma importancia para la construcción de las reglas difusas del selector. Dado que no siempre se consigue esta información de forma fácil se deben establecer atractivos que motiven al navegante a completar sus datos. Nuestro selector proporciona al navegante el atractivo de que la publicidad que éste recibirá será acorde a sus características y por ende de mayor interés para él.

Capítulo 4

Proceso de selección de las publicidades en el selector propuesto

4.1- Ingreso del usuario al sitio

Para que nuestro sistema pueda iniciar el proceso de selección de publicidades necesitamos contar con el perfil del visitante actual del sitio. Éste puede construirse a partir de datos reales del visitante actual o datos calculados sobre un promedio de la información de los visitantes registrados. Por esta razón se distinguen dos tipos de usuarios:

Usuario registrado: para el caso en que el usuario que ingrese al sitio se loguee previamente. En este caso hacemos uso de los datos que este ingresó al registrarse en el sitio.

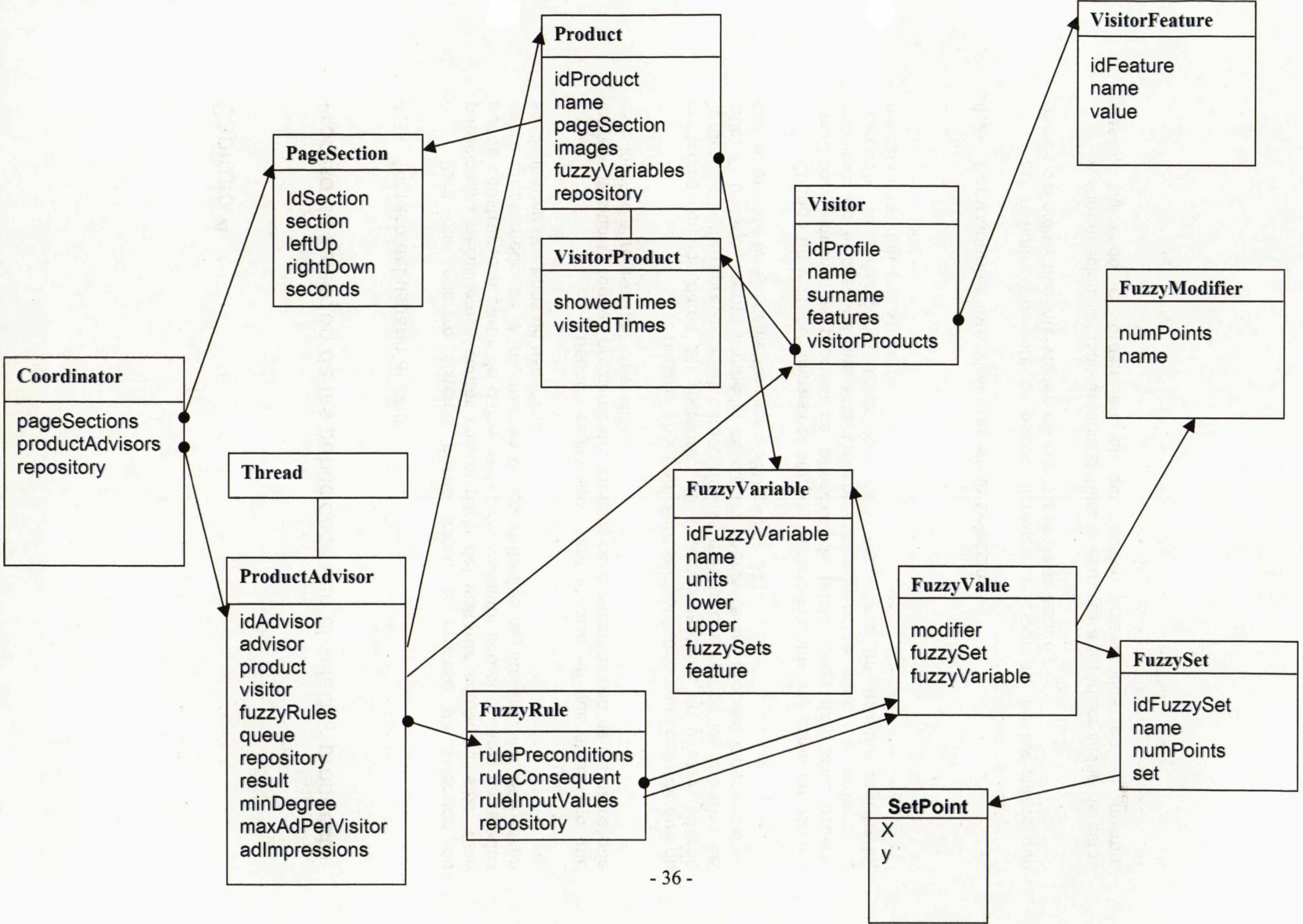
Usuario anónimo: Éste será utilizado en el caso de que no contemos con los datos del visitante. El sistema mantiene un usuario interno "promedio" que representa las características más frecuentes en los usuarios del sitio. El usuario interno promedio se calcula mediante un proceso independiente que se ejecuta periódicamente (ver SGBD página 31).

Cuando un visitante ingresa al sitio es tratado inicialmente como un usuario anónimo. Comienza el proceso de selección de publicidades para este usuario anónimo. En el caso de que este visitante se registre se reinicia el proceso de selección de publicidades tratándolo esta vez como un usuario registrado, utilizando los datos de su perfil.

4.2- Componentes intervinientes en la selección

En la figura siguiente se puede visualizar el diagrama de las clases que hemos definido y que intervienen en el proceso de selección.

A continuación del diagrama se detalla la estructura y funcionalidad de estas clases, siguiendo el orden en el que están presentadas en el mismo.


Product
idProduct
name
pageSection
images
fuzzyVariables
repository
VisitorFeature
idFeature
name
value
PageSection
IdSection
section
leftUp
rightDown
seconds
VisitorProduct
showedTimes
visitedTimes
Visitor
idProfile
name
surname
features
visitorProducts
FuzzyModifier
numPoints
name
Coordinator
pageSections
productAdvisors
repository
Thread
FuzzyVariable
idFuzzyVariable
name
units
lower
upper
fuzzySets
feature
ProductAdvisor
idAdvisor
advisor
product
visitor
fuzzyRules
queue
repository
result
minDegree
maxAdPerVisitor
adImpressions
FuzzyValue
modifier
fuzzySet
fuzzyVariable
FuzzySet
idFuzzySet
name
numPoints
set
FuzzyRule
rulePreconditions
ruleConsequent
ruleInputValues
repository
SetPoint
X
y

4.2.1- Coordinator

Esta clase modela el agente que coordina el proceso de selección de publicidades en el sitio. Entre los atributos se encuentran: una lista de productAdvisors (agentes auspiciantes activos), una lista de pageSections (secciones de la página para publicidad) y un repository (repositorio de datos).

Hay un único coordinador para cada visitante del sitio y se crea como consecuencia del ingreso del visitante al sitio. Su función es seleccionar cuáles son los anuncios más adecuados para mostrarle al visitante según su perfil, es decir, las características que se conocen del visitante.

Al inicializarse realiza las siguientes operaciones: crea el Repositorio de datos (para evitar accesos redundantes a la base de datos), carga las secciones activas, inicializa los agentes auspiciantes activos, obtiene los datos del usuario y registra en la base de datos la fecha y hora de entrada al sitio.

Luego comienza la selección de anuncios para el visitante actual, de la siguiente forma:

Cada agente auspiciante (ProductAdvisor) evalúa el posible interés del visitante en su producto o servicio mediante un "sistema de decisión de lógica difusa". El Coordinador mantiene una cola de auspiciantes ordenada en forma decreciente de acuerdo al resultado de cada una de estas evaluaciones. El coordinador le avisa a tantos agentes auspiciantes como secciones activas haya que muestre la publicidad de su producto en la sección dada. De esta forma se minimiza el desperdicio de anuncios.

Hay que destacar que la probabilidad de que se seleccione un anuncio cambia instantáneamente para cada visitante y que depende de la información disponible sobre él.

El código correspondiente a la clase Coordinator se encuentra en la pág. 87.

4.2.2- ProductAdvisor

Esta clase modela los agentes auspiciantes encargados de calcular el grado de adecuidad del producto que auspician para el visitante actual del sitio. Entre los atributos del productAdvisor figuran: un identificador, el nombre del auspiciante, el producto que se auspicia, el visitante actual del sitio, una lista de fuzzyRules, una cola (en la que se encolará este agente auspiciante una vez concluida su tarea), el repositorio de datos, result (variable numérica que almacena el grado de adecuidad calculado), el mínimo grado en que la publicidad debe adecuarse al visitante, la cantidad de impresiones contratadas, la cantidad máxima de veces que el anuncio será mostrado a cada visitante y las secciones de la página donde acepta que se muestre su publicidad. ProductAdvisor es subclase de la clase predefinida Thread.

La clase ProductAdvisor es nuclear. Incluye el Modelo Basado en Conocimiento Difuso asociado a un producto/servicio de un determinado anunciante. Existe un único ProductAdvisor para cada producto/servicio que se publicita en el sitio.

Definición: Un Modelo Basado en Conocimiento Difuso es una colección de n reglas de la forma:

If antecedente1 and antecedente2 and ... and antecedenteN then conclusion

Modelamos estas reglas con la clase FuzzyRule.

Es deseable que una persona con amplia experiencia en el negocio de este anunciante particular defina el Modelo Basado en Conocimiento Difuso asociado al producto / servicio que se publicita. Para obtener este modelo debe diseñar un conjunto de FuzzyRules (tantas como sean necesarias para considerar que se logró una buena descripción).

Ejemplo:

Si tenemos un anunciante que quiere ofrecer una crema para arrugas, el Modelo Basado en Conocimiento Difuso que éste puede definir es el constituido por las siguientes fuzzyRules:

If edad is extremadamente joven and sexo is masculino then Candidato is malo
If edad is extremadamente joven and sexo is femenino then Candidato is malo
If edad is muy joven and sexo is masculino then Candidato is malo
If edad is muy joven and sexo is femenino then Candidato is malo
If edad is joven and sexo is masculino then Candidato is malo
If edad is joven and sexo is femenino then Candidato is regular
If edad is media and sexo is masculino then Candidato is malo
If edad is media and sexo is femenino then Candidato is bueno
If edad is grande and sexo is femenino then Candidato is muy bueno
If edad is grande and sexo is masculino then Candidato is malo

El ProductAdvisor es el encargado de averiguar, utilizando un "sistema de decisión de lógica difusa" (ver pág. 24), el posible interés para el visitante actual del sitio en el producto/servicio que publicita. El "sistema de decisión de lógica difusa" razona acerca de los datos provistos por el Modelo Basado en Conocimiento Difuso y la información disponible sobre el visitante mediante el proceso general de inferencia. Este proceso consta de las siguientes etapas: Fusificación, Inferenciamiento (Inferencia y Composición) y Defusificación como hemos visto en la sección 2.1.12, pág. 24. El ProductAdvisor dirige este proceso general de inferencia:

Descripción del Proceso general de inferencia en nuestro selector

Fusificación: las funciones de pertenencia definidas sobre las variables de entrada (las cuales indican el grado en que un elemento pertenece a un conjunto), son aplicadas a los valores del visitante actual del sitio, para así determinar el valor de verdad para cada antecedente de la FuzzyRule.

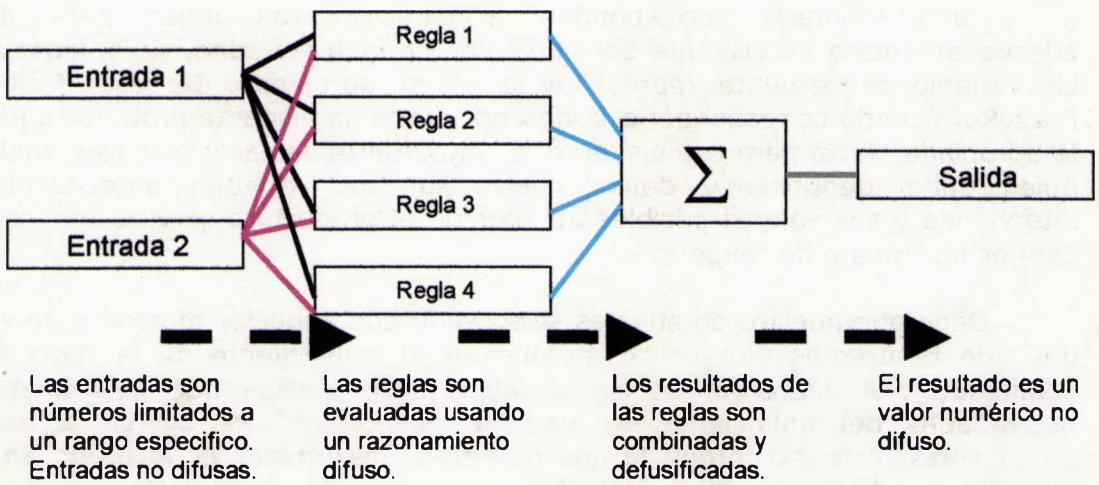
Inferencia: el ProductAdvisor se encarga de pedirle a cada FuzzyRule que realice su proceso de inferencia. Esto consiste en calcular el valor de verdad de los antecedentes de las FuzzyRules y aplicarlos a la parte de conclusión de las

FuzzyRules. Cada FuzzyRule se evalúa con los datos de la persona que en ese momento visita el sitio y como resultado obtiene un FuzzySet (un conjunto difuso).

Composición: la clase ProductAdvisor combina las salidas efectivas de cada una de las FuzzyRules obtenidas durante el proceso de Inferencia para lograr así la salida difusa efectiva del conjunto de FuzzyRules. Ésta agregación la realizamos tomando el punto máximo sobre todos los FuzzySets obtenidos de cada una de las FuzzyRules aplicadas con el método fuzzyUnion() de la clase FuzzySet.

Defusificación: convierte el FuzzySet resultante de la Composición en un número discreto. La clase ProductAdvisor le pide al FuzzySet que se defusifique mediante el método defussificate() de la clase FuzzySet. Existen distintos métodos para convertir un conjunto difuso en un número discreto como lo son el centro de área, el promedio pesado y el momento de máximo (ver pág. 23). Elegimos la del centro de área por ser la más comúnmente usada en lógica difusa. Este método es explicado en la clase FuzzySet.

A continuación mostramos un ejemplo gráfico para un sistema que consta de cuatro reglas difusas y recibe dos entradas.



La implementación correspondiente a la clase ProductAdvisor se encuentra en la pág. 89.

4.2.3- Fuzzy Rules

Esta clase modela las reglas difusas. Entre los atributos de una fuzzyRule se encuentran: rulePreconditions (lista de precondiciones), ruleConsequent (consecuente de la regla), ruleInputValues (valores del visitante actual del sitio que se usan para la evaluación de la fuzzyRule) y el repositorio de datos.

Las reglas en la lógica difusa son un modo de representar estrategias o técnicas apropiadas cuando el conocimiento proviene de la experiencia o de la intuición.

Son proposiciones que usan una condición del tipo IF-THEN (SI - ENTONCES):

IF <antecedente o condición> THEN <consecuente o conclusión>

El <antecedente> y el <consecuente> son Proposiciones Difusas que pueden formarse usando conjunciones (AND) o disyunciones (OR) en el sentido lógico usual como vimos en las págs 16 y 17.

Ejemplo: SI la Temperatura es Alta ENTONCES Abrir la válvula un poco

Estas reglas son relaciones que utilizamos para expresar el vínculo existente entre conceptos imprecisos, como podrían ser "edad es muy joven", "ingreso es algo alto", y el interés del visitante en el producto / servicio que se oferta. Estas reglas deben ser desarrolladas por un experto en el dominio en cuestión.

En nuestro trabajo las variables antecedentes (antecedente1, antecedente2, ... , antecedenteN) corresponden a características útiles para describir adecuadamente a un visitante del sitio como podrían ser edad, sexo, ingresos, etc. La variable consecuente representa la oferta apropiada. De esta forma cada FuzzyRule puede corresponder a la descripción de un visitante prototípico junto con la apropiada oferta para ese visitante. Es importante destacar que es el anunciante quien debe identificar y definir cuáles son las variables antecedentes que intervienen y sus valores posibles de acuerdo al producto o servicio que se oferte. Este es un trabajo de "experto".

Dado que nuestro objetivo es seleccionar qué anuncios mostrar a un visitante del sitio tendremos una única variable en el consecuente de la regla llamada "Candidato". A diferencia de las variables antecedentes que dependen de las necesidades del anunciante, la variable "Candidato" es común a todos los anunciantes. Esto es porque lo que queremos determinar es el grado en que el visitante puede ser un buen candidato para comprar el producto/servicio que se ofrece. Es la decisión de tener una única variable consecuente lo que determina que nuestro sistema sea un sistema "múltiple entrada única salida" [NFS 1996].

Los valores de entrada de la regla están dados por los datos correspondientes a la persona que visite el sitio. Estos datos están almacenados en la Base de Datos (ver Figura 2 pág. 24).

Es deseable que el formulario en el que los usuarios registran sus datos en un sitio debe ser diseñado de modo tal que exista una relación directa entre él y la información que se necesita para poder evaluar todas las FuzzyRules de todos los productos/servicios que se oferten.

Una FuzzyRule mantiene entonces un conjunto de FuzzyValues para representar los antecedentes de la regla, un único FuzzyValue que será el

consecuente de la regla (y se denomina Candidato) y otro conjunto para representar los valores de entrada de la regla.

La forma que tiene una FuzzyRule es:

If antecedente1 and antecedente2 and ... and antecedenteN then conclusión

Con más detalle:

If V_1 is A_{i1} and V_2 is A_{i2} and ... and y V_p is A_{ip} then U is B_i.

donde las V_i son las *variables* antecedentes y U es la variable consecuente. Las A_{ij}'s y las B_{ij}'s son *conjuntos difusos* sobre el correspondiente dominio de la variable.

Dado que U es la variable Candidato podemos aún mostrarla con más detalle y esta es la forma:

If V_1 is A_{i1} and V_2 is A_{i2} and ... and y V_p is A_{ip} then Candidato is B_i.

En nuestro sistema entonces se pueden encontrar reglas como las siguientes:

If edad is viejo and ingresos is medio then Candidato is bueno.

If estado civil is soltero and sexo is femenino then Candidato is muy bueno.

If nivel educativo is medio and cant. libros leídos is poco then Candidato is regular.

If grupo familiar is chico and cant. salidas semanales is muchas then Candidato is bueno

Mecanismo de inferencia:

Anteriormente explicamos que la clase ProductAdvisor dirige el proceso de inferencia, lo cual consiste en calcular el valor de verdad de los antecedentes de las FuzzyRules y aplicarlos a la parte de conclusión de las mismas. Cada FuzzyRule es evaluada con los datos de la persona que en ese momento visita el sitio y como resultado se obtiene un FuzzySet.

Realizar el proceso de inferencia de una regla es calcular la salida efectiva de la misma. Decidimos que a este cálculo lo efectuaremos escalando el consecuente (también llamado conclusión) de la regla, por el "nivel de disparo" (grado de verdad para la premisa de la regla) de ésta.

- **Obtención del nivel de disparo de una FuzzyRule:** A continuación describimos los pasos para obtener el nivel de disparo de una FuzzyRule:

1- Evaluamos cada uno de los antecedentes utilizando el perfil del usuario (datos correspondientes a la persona que en ese momento visite el sitio). Esto es lo que en lógica difusa se denomina fusificar las entradas y consiste en aplicar las

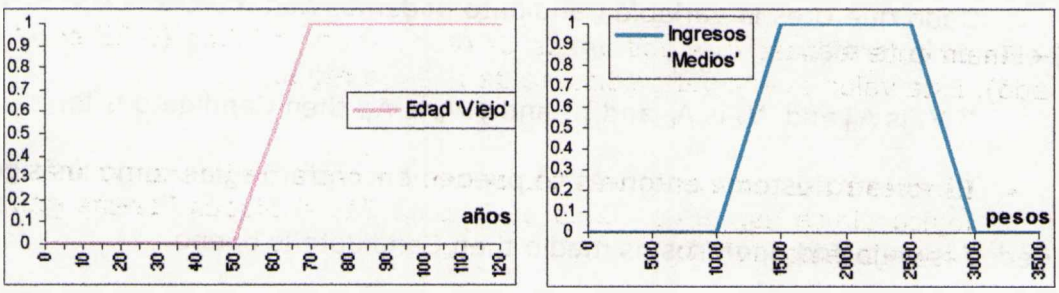
funciones de pertenencia definidas sobre las variables de entrada a los valores del visitante actual del sitio, para así determinar el valor de verdad para cada antecedente de la regla.

Ejemplo: Fusificación de las entradas

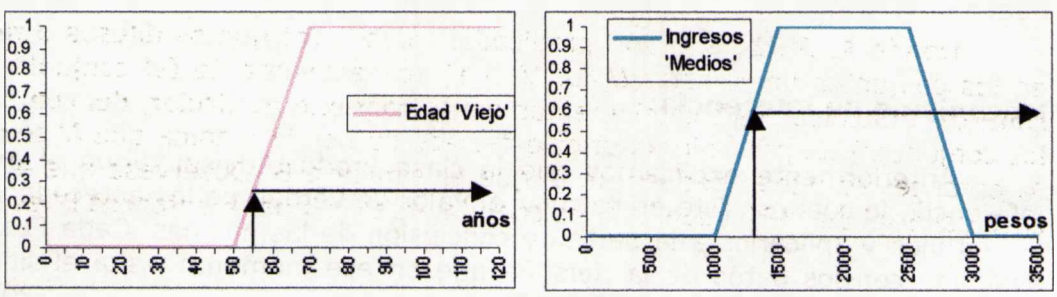
Sea la siguiente FuzzyRule:

If edad is viejo and ingresos is medio then Candidato is bueno.

Sean las funciones de pertenencia definidas para los conjuntos difusos "Medio" y "Viejo":



Supongamos que nuestro visitante tiene 55 años y su ingreso es de 1300 pesos. Evaluamos 55 con el conjunto difuso "viejo" y obtenemos 0.25. Evaluamos 1300 con el conjunto difuso "medio" y obtenemos 0.6.



Así queda establecido en que grado esta persona es "vieja" y en que grado su salario es "medio".

2- A continuación aplicamos el operador difuso "and" en el antecedente. Dado que le brindamos al anunciante la posibilidad construir FuzzyRules con antecedentes compuestos para lograr una mejor descripción, necesitamos una t-norma para definir el operador de conjunción. Hemos dicho que una t-norma representa la intersección de dos conjuntos difusos (ver pág. 10). Los dos operadores más conocidos son el mínimo y el producto. Elegimos el primero por ser un operador robusto y fácilmente implementable. Es la elección tradicionalmente elegida en lógica difusa para realizar la conjunción. El operador mínimo calcula el mínimo numérico entre los resultados del paso anterior, es decir, luego de evaluar cada uno

de los antecedentes con los datos del visitante del sitio se queda con el de valor menor.

Ejemplo: Aplicar "and" para obtener el nivel de disparo de una FuzzyRule

If edad is viejo and ingresos is medio then Candidato is bueno

Con un visitante con salario de 1300 pesos de 55 años de edad.

Ya que el resultado de evaluar 55 con el conjunto difuso "viejo" es 0.25 y el resultado de evaluar 1300 con el conjunto difuso "medio" es 0.6, entonces nuestro operador mínimo se obtendrá realizando min(0.25, 0.6) dando como resultado 0.25.

3- Finalmente el paso que nos queda es retornar ese mínimo (0.25 en el ejemplo dado). Este valor es el nivel de disparo de nuestra regla.

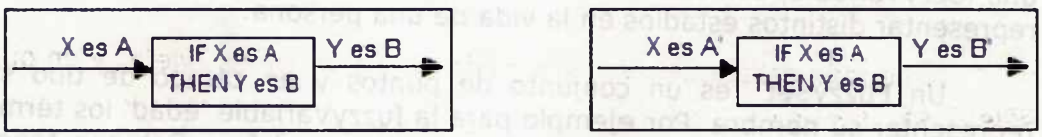
- **Elección del proceso de inferencia:** La regla de inferencia más usada en lógica difusa es, como vimos en la pág. 21, el Modus Ponens generalizado (abreviaremos MPG):

  Premisa 1: IF x is A, THEN y is B

  Premisa 2: x is A'

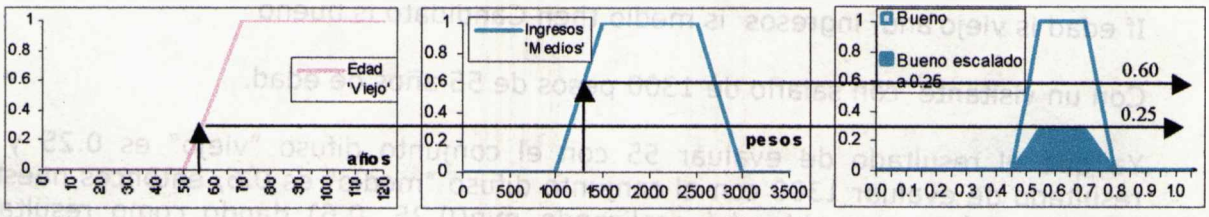
  Conclusion: y is B'.

Donde A, A', B, y B' son predicados difusos (conjuntos difusos o relaciones) en los conjuntos universales *U, U, V,* y *V,* respectivamente (el conjunto universal contiene todos los posibles elementos en un contexto particular, del cual se forman los conjuntos). El principio anterior establece que en la medida que A' sea similar a A, entonces se puede deducir un B' que es similar a B. Obsérvese que el MP es un caso especial del MPG cuando A' es completamente igual A entonces B' es B.



Una regla difusa es expresada como una implicación A ⇒ B (If antecedente then conclusión). Hemos elegido la operación producto de Larsen para implementar la implicación, la cual se obtiene si el producto algebraico es usado en la conjunción, multiplicando el nivel de disparo de la regla por la conclusión de la misma. Esta elección se debe a que observamos que es una de las más frecuentemente usadas en reglas de inferencia difusa por su simplicidad computacional y eficiencia [NFS 1996]. Si se compara el método de Larsen con otras reglas de implicación difusa se ve que es una de las más adecuadas para el Modus Ponens Generalizado (ver pág. 22).

El siguiente gráfico muestra el fuzzySet obtenido como resultado de aplicar el "mecanismo de inferencia" a la regla vista.



El resultado de evaluar 55 con el fuzzySet "Viejo" es 0.25
El resultado de evaluar 1300 con el fuzzySet "Medios" es 0.60
El nivel de disparo de la regla es min(0.15, 0.60)=0.25
La salida efectiva de la regla es escalar el fuzzySet "Bueno" a 0.25, esto es, multiplicar cada valor de pertenecia de "Bueno" por 0.25.

El código correspondiente a la clase FuzzyRule se encuentra en la pág. 92.

4.2.4- FuzzyValue

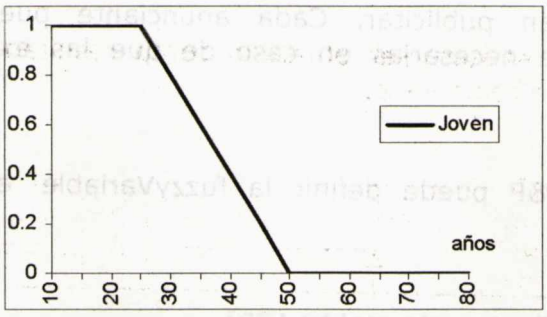
Es una entidad que utilizamos para la construcción de las FuzzyRules. Los antecedentes de una FuzzyRule se representan mediante una lista de FuzzyValues y un único FuzzyValue será el consecuente de la FuzzyRule.

Un FuzzyValue está constituido por una FuzzyVariable, un FuzzySet y un FuzzyModifier.

La FuzzyVariable proporciona un conjunto de términos que pueden ser usados para describir el concepto al cual la variable se refiere. Por ejemplo, edad puede ser una fuzzyVariable, con términos tales como niño, joven, mediana edad y viejo para representar distintos estadios en la vida de una persona.

Un FuzzySet es un conjunto de puntos y un campo de tipo string para representar su nombre. Por ejemplo para la fuzzyVariable 'edad' los términos antes mencionados niño, joven,...etc serán fuzzySets y el fuzzySet que tenga nombre 'niño' tendrá un conjunto de puntos asociado que representará el concepto niño. Lo mismo ocurre para el fuzzySet "joven" y para los demás términos que compongan la fuzzyVariable "edad".

Ejemplo: conjunto difuso "joven"



Un fuzzyModifier es un operador para modificar el significado de un conjunto difuso A para crear un nuevo conjunto difuso h(A) (como hemos visto en la sección 2.1 en la pág. 16) Por ejemplo en el conjunto difuso "MUY joven", "MUY" es un modificador. Proveemos en nuestro selector un amplio número de modificadores entre los cuales están LIGERAMENTE, MAS O MENOS, NO, EXTREMADAMENTE y para cuando no se requiera un modificador usaremos el modificador NINGUNO el cual al ser aplicado sobre un fuzzySet lo deja tal cual está. Cuando se inicializa un fuzzyValue se aplica el modificador al fuzzySet para evitar realizar este paso en cada uno de los cálculos que sea necesario hacer.

El código correspondiente a la clase FuzzyValue se encuentra en la pág. 94.

4.2.5- FuzzyVariable

Esta clase modela una variable difusa. Define el lenguaje que será usado para tratar un concepto difuso tal como temperatura, presión, edad, etc.

Dicho objeto tiene los siguientes elementos:
- Un **nombre** (*temperatura, edad*), mediante un campo de tipo *String*.
- Las **unidades** de la variable (*grados, años*), también como un campo de tipo *String*.
- El **universo de discurso** de la variable. El valor mínimo para esa unidad (por ejemplo 0), el valor máximo para esa unidad (por ejemplo 130).
- Una lista de **fuzzySets** que serán usados para describir la variable difusa.

Hay variables cuya definición es más compleja porque se mueven en dominios poco claros y por ello puede no ser sencillo trasladarlos a valores numéricos. Por ejemplo: "limpieza", "sabiduría" y "verdor".

Ejemplo:
Nombre: "sexo"
Unidad: -
Universo de discurso: el intervalo entero [0,1]
Lista de fuzzySets: "femenino" y "masculino".

El sistema permite que cada sitio predefina las fuzzyVariables que considere que puedan ser de mayor utilidad. Estas fuzzyVariables pueden ser usadas por los anunciantes para crear los antecedentes de las fuzzyRules asociadas al

producto/servicio que desean publicitar. Cada anunciante puede definir las fuzzyVariables que considere necesarias en caso de que las existentes no se adapten a sus necesidades.

Ejemplo:

El anunciante P&P puede definir la fuzzyVariable 'edadP&P' de la siguiente manera:
Nombre: "edadP&P"
Unidad: años
Universo de discurso: El intervalo real [0,120]
Lista de fuzzySets: "niño", "joven", "media" y "viejo".

Todas las fuzzyRules asociadas a todos los productos/servicios tienen la misma variable en el consecuente denominada "Candidato". Esta variable es provista por nuestro sistema y no puede ser modificada porque como ya hemos dicho para todos los anunciantes tiene el mismo significado.

Definimos la variable Candidato de la siguiente manera:

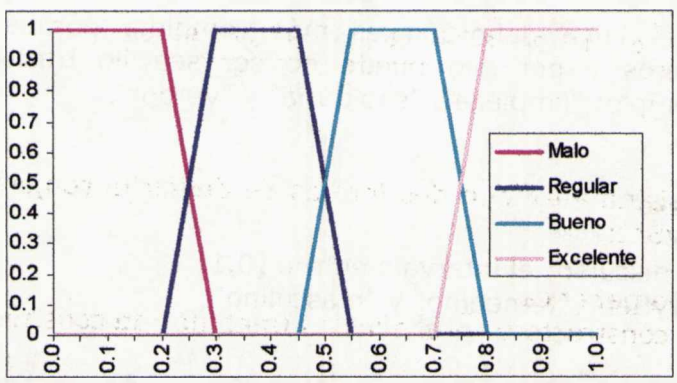
Nombre: "Candidato"
Unidad: frecuencia de clickeo
Universo de discurso: El intervalo real [0,1]
Lista de fuzzySets: "Excelente", "Bueno", "Regular" y "Malo".

La frecuencia de clickeo (FC) la definimos como:

$$FC = \frac{CAC}{CAM}$$

donde CAC es la cantidad de anuncios en los que el visitante clickeó y CAM es la cantidad total de anuncios mostrados al visitante.

Graficamos los fuzzySets correspondientes a la variable "Candidato":



El código correspondiente a la clase FuzzyVariable se encuentra en la pág. 96.

4.2.6- FuzzySet

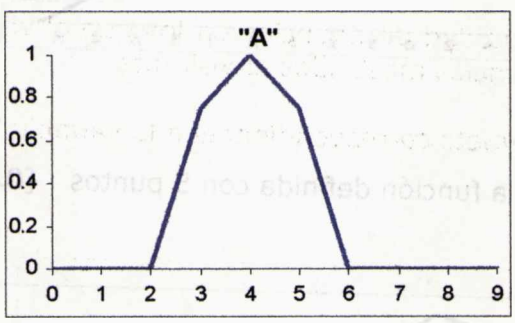
Representamos esta clase mediante:

- un **nombre** (por ejemplo *caliente, templado, frío*).
- un **conjunto de puntos** que define su función de pertenencia para los elementos del universo de discurso.

El nombre de un fuzzySet se representa mediante un string. Por ejemplo para la fuzzyVariable edad podemos tener los fuzzySets niño, joven,...etc y el fuzzySet que tenga nombre joven tendrá un conjunto de puntos asociado que representará el concepto joven.

Como hemos visto en la sección 2.1.1, en el conjunto de puntos μ/x, μ es el valor de pertenencia para el número real x manejando así un conjunto de valores tales como $\{ \mu_1/x_1 \ \mu_2/x_2 \ldots \mu_n/x_n \}$. El valor de pertenencia μ cae en el rango [0,1] aunque en algunos casos es posible que este valor sea mayor a 1.0: por ejemplo cuando se realiza una suma de valores difusos. Los valores x están ordenados en el conjunto de forma creciente de forma que (x1 < = x2 <=... <= xn). Los valores previos a x_1 tienen el mismo valor de pertenencia que x_1 y los valores posteriores a x_n tienen el mismo valor de pertenencia que x_n . Los valores entre x_l y x_{l+1} son determinados por la línea que une estos dos puntos consecutivos.

Por ejemplo el conjunto difuso "A" { 0.0/2.0, 0.75/3.0, 1.0/4.0, 0.75/5.0, 0.0/6.0} representa un trapecio extendido y con este conocimiento podemos determinar el valor que toma este conjunto para los valores 1.5, 3.5 etc. Para representar un singleton (un conjunto donde sólo valga 1 un determinado valor, y 0 el resto), tendríamos que hacer {0.0 / 0.5, 1.0 / 0.5, 0.0 / 0.5}.



En nuestro sistema existen dos formas de definir el conjunto de puntos que constituye un fuzzySet:

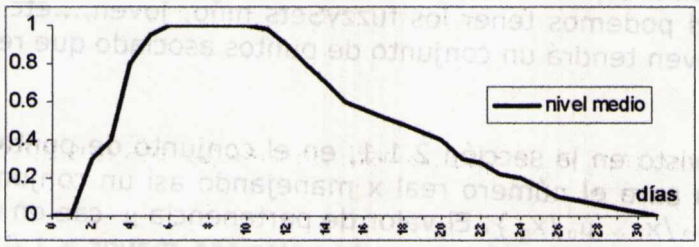
- La forma genérica
- Uso de constructores predefinidos (métodos de construcción)

En la forma genérica se requiere especificar cada uno de los puntos que darán forma al conjunto, como si fuese una definición "por extensión". Como el valor de pertenencia para los puntos intermedios entre los puntos dados se calcula

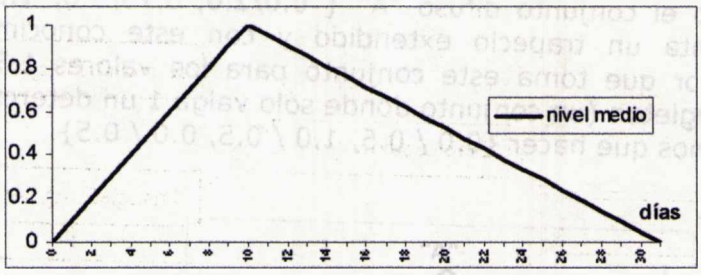
mediante la recta que los une, en la medida en que se provean más cantidad de puntos la forma de la función se ajustará mas a la deseada.

Ejemplo:

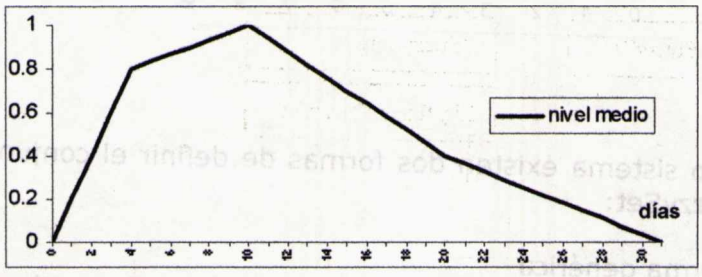
Supongamos que un anunciante quiere representar para la fuzzyVariable "nivel de compras durante el mes" el concepto "nivel medio" mediante la siguiente función:



Vemos cómo se ve la función definida con 3 puntos {0.0 / 1, 1.0 / 10, 0.0 / 31}.

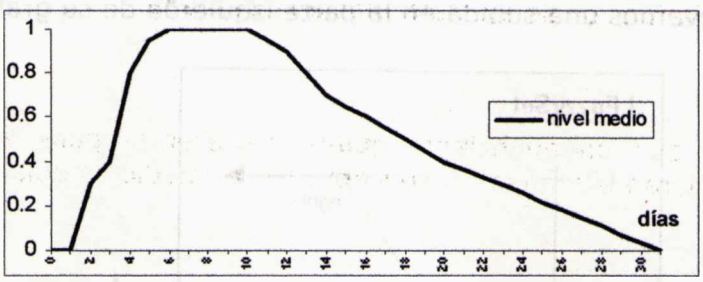


Vemos cómo se ve la función definida con 5 puntos {0.0 / 1, 0.8/4, 1.0 / 10, 0.4/20, 0.0 / 31}.



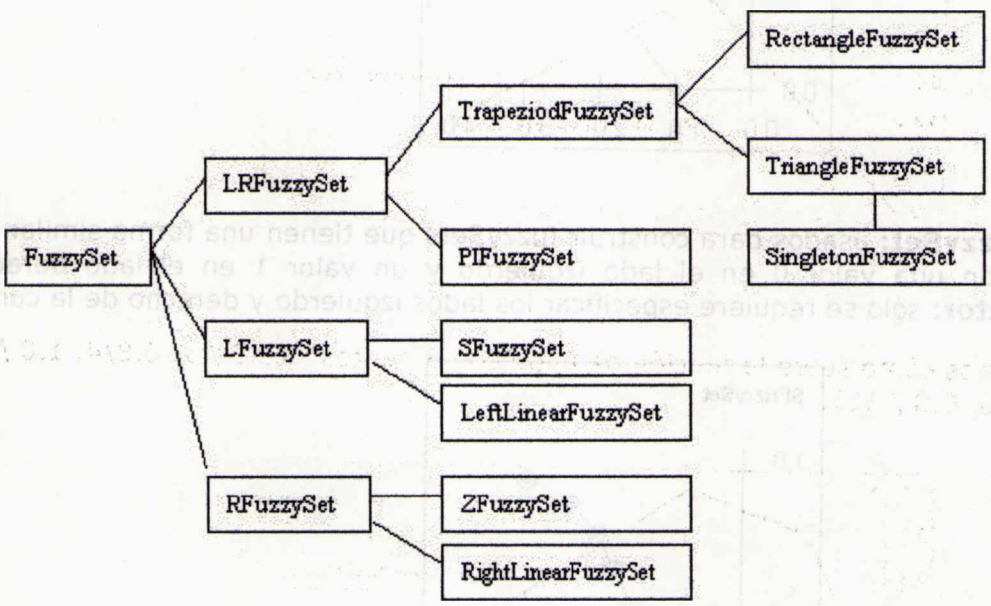
Por último vemos cómo se ve la función si se define con 15 puntos.

{0.0/1, 0.3/2, 0.4/3, 0.8/4, 0.95/5, 1.0/6, 1.0/7, 1.0/8, 1.0/9, 1.0/10, 0.9/12, 0.7/14, 0.6/16, 0.4/20, 0.0/31}.



Se observa cómo a medida que se definen más puntos nos aproximamos más a la forma de la función original.

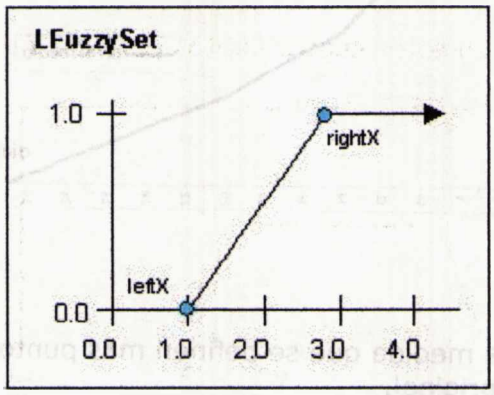
El uso de constructores predefinidos facilita la creación y comprensión de conjuntos difusos. Consideramos útil definir una jerarquía de subclases. Esta jerarquía permite satisfacer todos los posibles requerimientos de los anunciantes al momento de definir los términos de las variables difusas. La jerarquía que hemos definido es la siguiente:



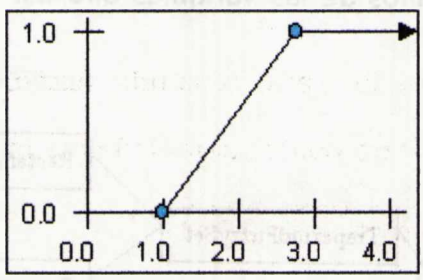
A continuación describimos las clases de la jerarquía que hemos definido:

A- LFuzzySet: se usa para construir fuzzySets especializados que en general tienen valores de pertenencia 0 en la arista izquierda. No es común usar directamente esta clase para crear fuzzySets, sino que se usan sus subclases. La

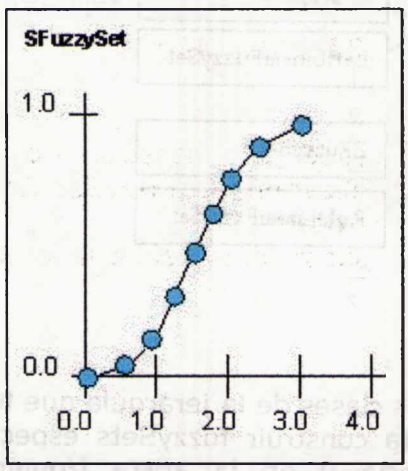
forma de la curva entre los valores leftX y righX son determinados por la función especificada, observamos una subida en la parte izquierda de su gráfica.



A.1- LeftLinearFuzzySet: se usan para construir un lFuzzySets que tienen una línea derecha del valor 0 del lado izquierdo menor al valor 1 del lado derecho superior.



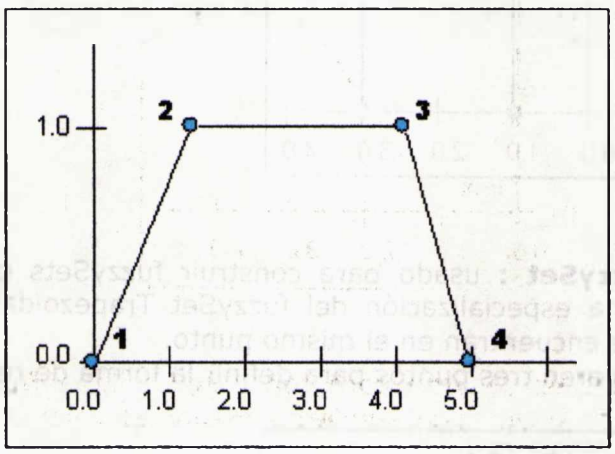
A.2- SFuzzySet: usados para construir fuzzySets que tienen una forma similar a la letra S con una valor 0 en el lado izquierdo y un valor 1 en el lado derecho. **Constructor:** sólo se requiere especificar los lados izquierdo y derecho de la curva.



B- LRFuzzySet: permite la construcción de un rango variado de fuzzySets con una forma similar. El punto **1** es el más izquierdo que tiene un valor de pertenencia 0. El **2** es el punto más izquierdo con valor de pertenencia 1. El punto **3** es el punto de

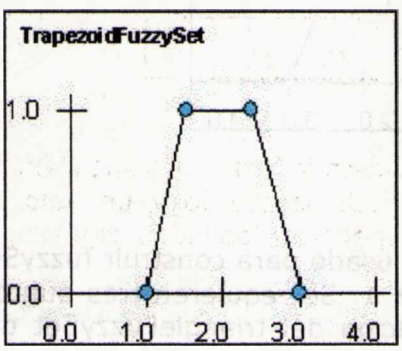
más a la derecha con valor de pertenencia 1 y por último el punto **4** es el de más a la derecha con valor de pertenencia 0. El IRFuzzySet será siempre normal y convexo, así como la función de pertenencia de los puntos de inicio y fin serán siempre 0.

Constructor: se requieren estos cuatro puntos para construir el fuzzySet.



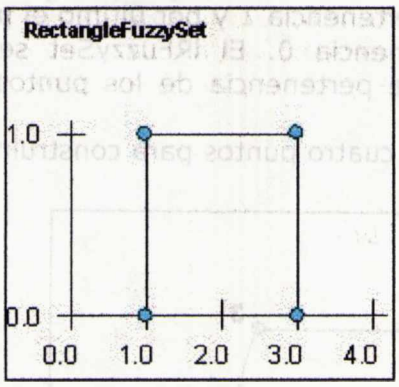
B.1-TrapezoidFuzzySet: usado para construir fuzzySets que poseen forma trapezoidal.

Constructor: Se requieren cuatro puntos para definir la forma de trapezoide.

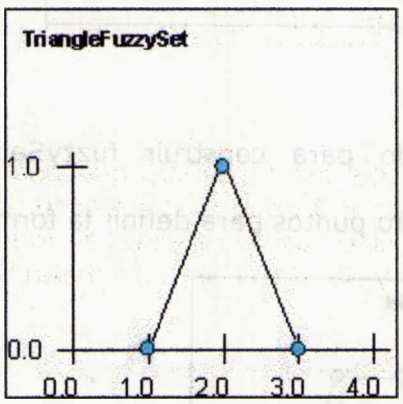


B.1.1- RectangleFuzzySet: usado para construir fuzzySets que poseen forma rectangular. Esta es una especialización del fuzzySet Trapezoidal donde los lados izquierdo y derecho son lineas verticales.

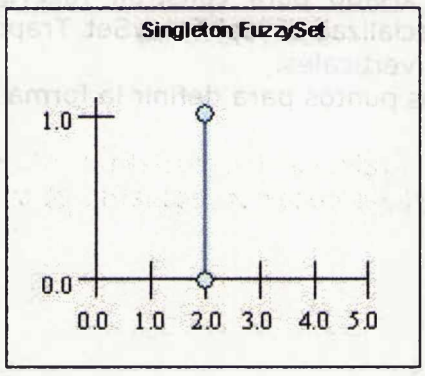
Constructor: Se requieren dos puntos para definir la forma de rectángulo.



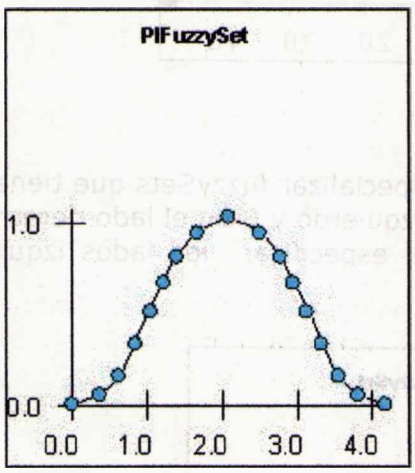
B.1.2- Triangle FuzzySet : usado para construir fuzzySets que poseen forma triangular. Esta es una especialización del fuzzySet Trapezoidal donde los lados izquierdo y derecho se encuentran en el mismo punto.
Constructor: Se requieren tres puntos para definir la forma de rectángulo.



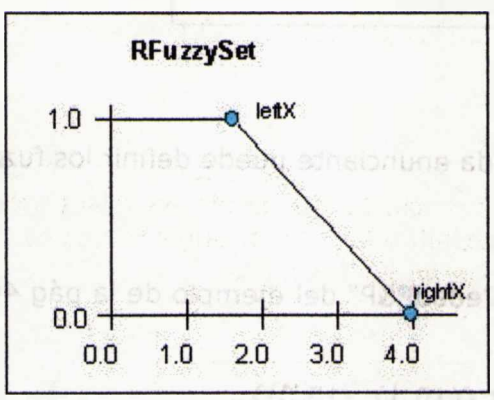
B.1.2.1- Singleton FuzzySet: usado para construir fuzzySets que tienen un valor simple con valor de pertenencia 1. Se requieren tres puntos para definirlo: (x,0), (x,1), (x,0). Es una especialización del triangleFuzzySet donde los valores de la izquierda, medio y derecha son los mismos.
Constructor: requiere de un solo valor para definirlo.



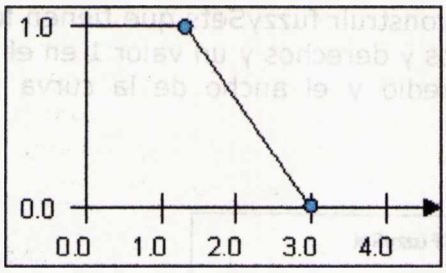
B.2- PIFuzzySet: usado para construir fuzzySets que tienen forma de campana con un valor 0 en los lados izquierdos y derechos y un valor 1 en el medio.
Constructor: sólo el punto medio y el ancho de la curva son los datos que se necesitan especificar.



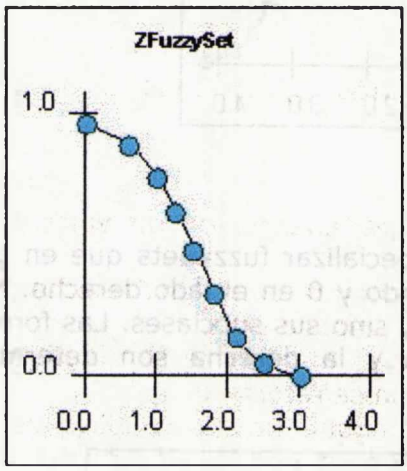
C- RFuzzySet: usado para especializar fuzzySets que en general tienen valor de pertenencia 1 en el lado izquierdo y 0 en el lado derecho. No se usa directamente esta clase para crear fuzzySets, sino sus subclases. Las formas de las curvas entre los valores x de la izquierda y la derecha son determinados por la función especificada.



C.1- RightLinearFuzzySet: usado para especializar rFuzzySets que tienen una línea derecha del valor 1 del lado superior izquierdo al valor 0 del lado superior derecho.



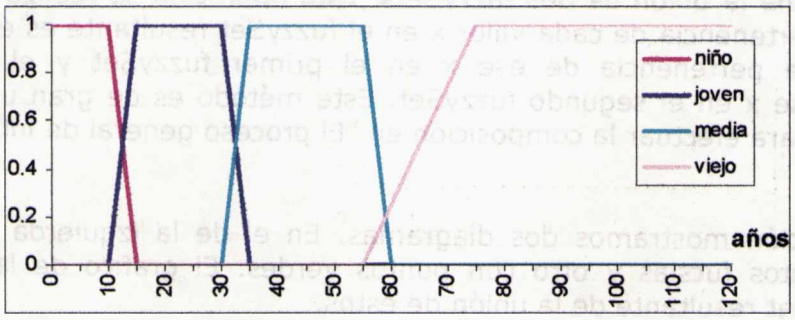
C.2- ZFuzzySet: usado para especializar fuzzySets que tienen una forma similar a la letra z con valor 1 en el lado izquierdo y 0 en el lado derecho.
Constructor: sólo se necesitan especificar los lados izquierdo y derecho de la curva.



En nuestro sistema cada anunciante puede definir los fuzzySets de acuerdo a sus necesidades.

Ejemplo:
Para la fuzzyVariable "edadP&P" del ejemplo de la pág 46) los fuzzySets podrían estar definidos como:

1. niño: un rFuzzySet {(10,1), (15,0)};
2. joven: un trapezoidFuzzySet {(10,0), (15,1), (30,1), (35,0)};
3. media: un trapezoidFuzzySet {(30,0), (35,1), (55,1), (65,0)}
4. viejo: un lFuzzySet {(55,0), (75,1)};



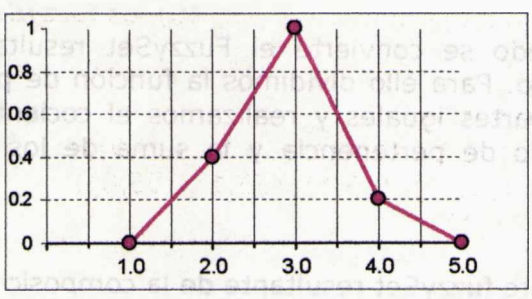
En el gráfico anterior se puede observar que un elemento puede pertenecer parcialmente a un conjunto y pertenecer parcialmente a otro conjunto a la vez, como se mencionó en la sección 2.1.1. Vemos cómo, por ejemplo, una persona de 61 años es considerada "media" con valor de pertenencia 0.6 y "vieja" con valor de pertenencia 0.25.

A continuación escribimos tres operaciones sobre fuzzySets de especial importancia para nuestro selector:

evaluate: Retorna el valor de pertenencia de un valor x dado. Hay tres casos posibles:

- Caso 1: si el valor x se encuentra entre los definidos por extensión se retorna el valor de pertenencia correspondiente.
- Caso 2: si el valor x dado es menor que el mínimo o mayor que el máximo valor definido por extensión se retorna 0.
- Caso 3: si no se dan ninguna de las condiciones anteriores se halla la ecuación de la recta que pasa por los dos puntos definidos por extensión consecutivos entre los que se encuentra el x dado.

Por ejemplo, dado el fuzzySet {0.0/1.0, 0.4/2.0, 1.0/3.0, 0.2/4.0, 0.0/5.0} que se puede representar mediante la siguiente figura:

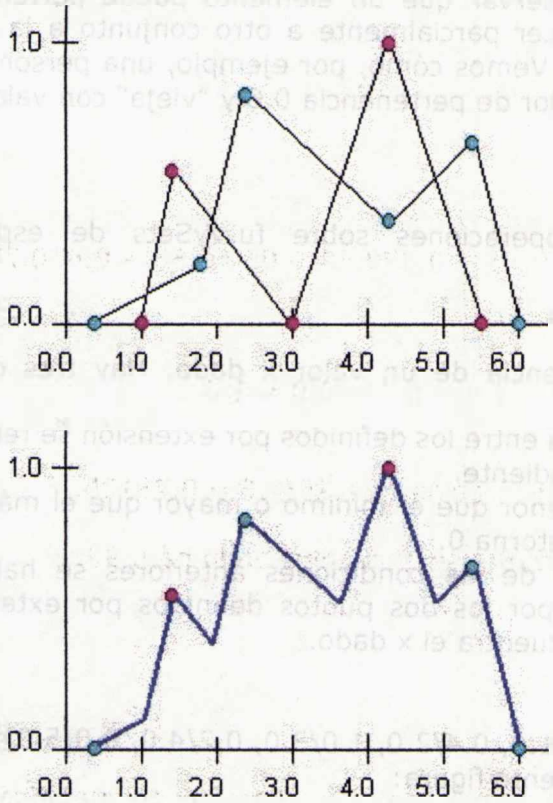


(Caso 1:) dado x=2.0, evaluate(x)=0.4
(Caso 2:) dado x=1.0 o x=5.0, evaluate(x)=0.0
(Caso 3:) dado x=3.5, evaluate(x)=0.6

fuzzyUnion: Retorna la unión de dos fuzzySets. Ésta operación se realiza de modo que el valor de pertenencia de cada valor x en el fuzzySet resultante es el máximo entre el valor de pertenencia de ese x en el primer fuzzySet y el valor de pertenencia de ese x en el segundo fuzzySet. Este método es de gran utilidad en nuestro selector para efectuar la composición en "El proceso general de inferencia".

Ejemplo

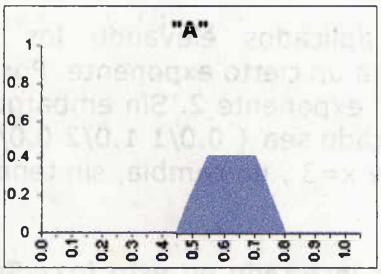
A continuación mostramos dos diagramas. En el de la izquierda se ve un fuzzySet con puntos fucsias y otro con puntos verdes. El grafico de la derecha muestra el fuzzySet resultante de la unión de éstos.



defussificate: Mediante éste método se convierte el FuzzySet resultante de la Composición en un número discreto. Para ello dividimos la función de pertenencia correspondiente al fuzzySet en partes iguales y realizamos el cociente entre la suma de cada punto por su grado de pertenencia y la suma de los grados de pertenencias.

Ejemplo:

Supongamos tener el siguiente fuzzySet resultante de la composición sobre la variable Candidato:



Recordemos que:

$$Z^*_{CDA} = \frac{\sum_{j=1}^{n} z_j \, \mu_C(z_j)}{\sum_{j=1}^{n} \mu_C(z_j)}$$

entonces tenemos:

$\sum_{j=1}^{n} z_j \, \mu_C(z_j) = 0.0*0.45 + 0.2*0.5 + 0.4*0.55 + 0.4*0.60 + 0.4*0.65 + 0.4*0.70 + 0.2*0.75 + 0.0*0.8 = 1.25$

y:

$\sum_{j=1}^{n} \mu_C(z_j) = 0.0 + 0.2 + 0.4 + 0.4 + 0.4 + 0.4 + 0.2 + 0.0 = 1.6$

Por lo tanto el valor obtenido por la defusificación es 1.25/1.6 = 0.78125.

El código correspondiente a la clase FuzzySet se encuentra en la pág. 99.

4.2.7- FuzzyModifiers

Esta es una "clase abstracta" que contiene una jerarquía de modificadores disponible para operar sobre conjuntos difusos.

Se emplean modificadores para aumentar la posibilidad de logrgar buenas descripciones, modificando la forma de los fuzzySets asociados a las fuzzyVariables.

Como hemos comentado en la pág. 16 un modificador puede ser usado para mejorar la habilidad para describir conceptos difusos. Los modificadores tales como MUY, LIGERAMENTE son usados en frases como "muy joven", "ligeramente viejo" y en lógica difusa cambian la forma, al modificar los valores de pertenencia, de un conjunto difuso dado (ver gráficos explicativos en la siguiente página). Cada modificador proporciona un método llamado "call" que acepta como parámetro un fuzzySet y retorna el fuzzySet resultante de la aplicación de este modificador al fuzzySet dado. El tipo de la operación podemos describirlo como:

"call": FuzzySet → FuzzySet

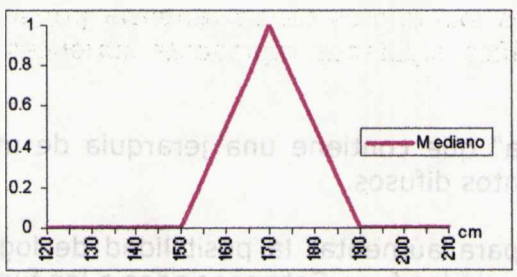
Muchos de los modificadores son aplicados elevando los valores de pertenencia de los puntos del conjunto difuso a un cierto exponente. Por ejemplo, el modificador MUY es implementado usando el exponente 2. Sin embargo, considere el caso donde el conjunto difuso a ser modificado sea { 0.0/1 1.0/2 0.0/3}. En este caso, el valor de pertenencia para x=1, x=2 y x=3 , no cambia, sin tener en cuenta el valor del exponente.

Sin embargo, si un punto adicional es insertado en este fuzzySet de forma que el conjunto original pase a ser { 0.0/1 0.5/2.5 1.0/2 0.0/3}, y luego el modificador MUY es aplicado, el punto que era originalmente 0.5/2 será ahora 0.25/2.5. Si nosotros insertamos otros puntos adicionales en valores de x arbitrarios encontraremos que a menos que sus valores de pertenencia sean 0 ó 1, éste cambiará al aplicarse el exponente.

Debido a que la representación de la función de pertenencia que dimos es un conjunto de puntos que son unidos por rectas, encontramos que puede ser necesario agregar puntos adicionales al conjunto difuso para mantener la precisión en el conjunto difuso resultante. Los puntos son agregados al fuzzySet a una cierta distancia *delta* uno de otro y antes de la aplicación del modificador. Este *delta* es calculado en función de los valores máximo y mínimo de x y de la cantidad de puntos que forman el conjunto. A continuación se calcula el valor de pertenencia que toma la función para cada uno de estos nuevos puntos.

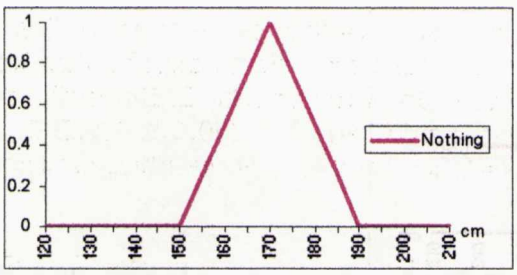
Seguidamente mostramos los modificadores que proveemos en el selector.

Supongamos tener el siguiente conjunto que representa altura "mediano" visto en la pág. 6:

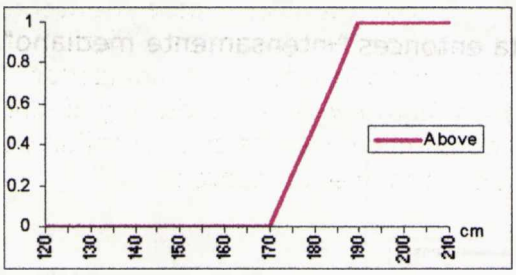


Se mostrarán los gráficos de los modificadores aplicados a este conjunto en particular.

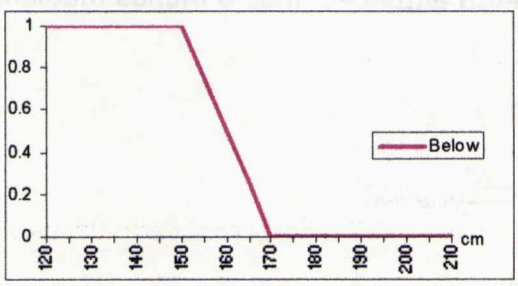
Ninguno (Nothing): Este modificador se usa cuando no se desea usar ninguno de los modificadores, su efecto es nulo ya que el fuzzySet pasado como parámetro no recibe cambios.



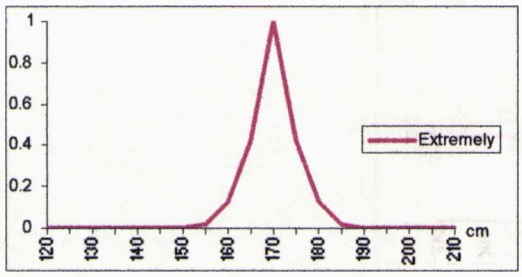
Arriba de (Above): Este modificador identifica el primer valor de x al cual el máximo valor es alcanzado. Todos los valores de pertenencia por debajo de este punto son seteados a cero y todos los valores de pertenencia por arriba de este valor son seteados a 1-y. El gráfico siguiente representa entonces "arriba de mediano".



Debajo de (Below): Este modificador identifica el primer valor de x al cual el máximo valor es alcanzado. Todos los valores de pertenencia por arriba de este punto son seteados a cero y todos los valores de pertenencia por abajo de este valor son seteados a 1-y. El gráfico siguiente representa entonces "debajo de mediano".



Extremadamente (Extremely): Este modificador expande el fuzzySet pasado como parámetro y luego eleva todos los valores de pertenencia del fuzzySet a un factor de 3. El gráfico siguiente representa entonces "extremadamente mediano".

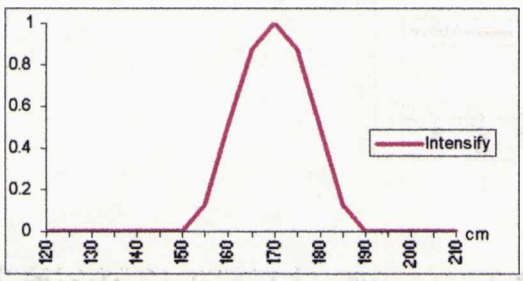


Intensificador (Intensify): Este modificador expande el FuzzySet pasado como parámetro y luego realiza las siguientes modificaciones sobre éste:

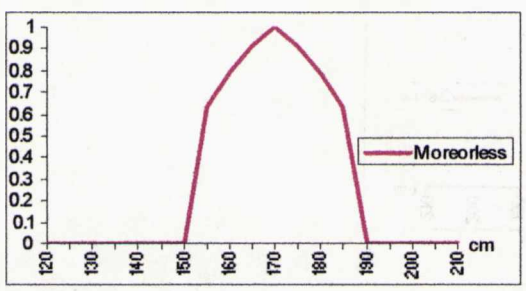
- si(0.0<=y<=0.5) entonces y= 2*y^2
- si(0.5<y<=1.0) entonces y= 1-2*(1-y)^2

Lo cual tiene el efecto de disminuir los valores menores que 0.5 y aumentar los otros.

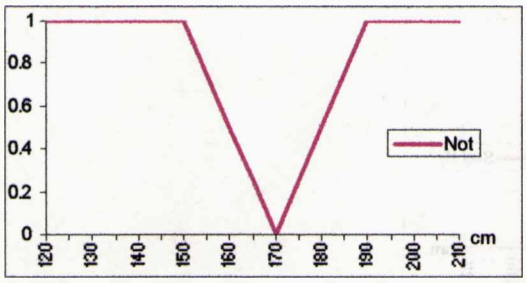
El gráfico siguiente representa entonces "intensamente mediano".



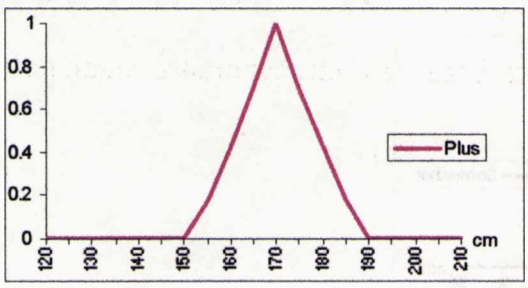
Más o menos (Moreorless): Este modificador expande el FuzzySet pasado como parámetro y luego eleva todos los valores de pertenencia del FuzzySet a un factor de 1/3. El gráfico siguiente representa entonces "más o menos mediano".



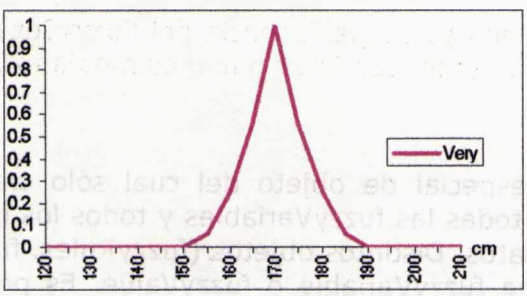
No (Not): Este modificador retorna el complemento del FuzzySet pasado como parámetro. El gráfico siguiente representa entonces "no mediano".



Más (Plus): Este modificador expande el FuzzySet pasado como parámetro y luego eleva todos los valores de pertenencia del FuzzySet a un factor de 1.25. El gráfico siguiente representa entonces "más mediano".



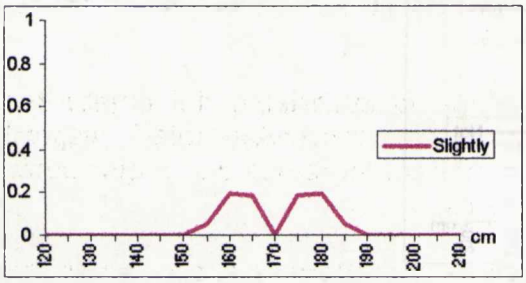
Muy (Very): Este modificador expande el FuzzySet pasado como parámetro y luego eleva todos los valores de pertenencia del FuzzySet a un factor de 2. El gráfico siguiente representa entonces "muy mediano".



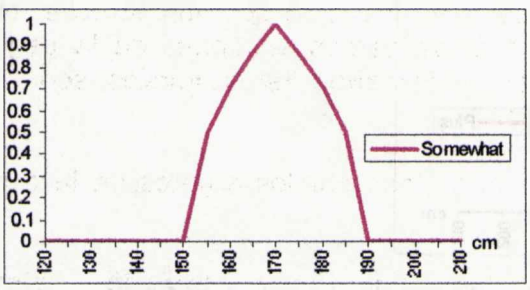
Ligeramente (Slightly): Este modificador expande el FuzzySet pasado como parámetro y luego realiza las siguientes modificaciones sobre éste:

- intensify[plus(A) AND not(very(A)))]

El gráfico siguiente representa entonces "ligeramente mediano".



Algo (Somewhat): Este modificador expande el FuzzySet pasado como parámetro y luego eleva todos los valores de pertenencia del FuzzySet a un factor de 1/2. El gráfico siguiente representa entonces "algo mediano".



Vemos luego de ejemplificar, cómo logramos variantes en la descripción de la idea de "mediano".

El código correspondiente a la clase FuzzyModifier se encuentra en la pág. 116.

4.2.8- Repository

Esta clase modela un tipo especial de objeto del cual sólo creamos una instancia. El objetivo es almacenar todas las fuzzyVariables y todos los fuzzyValues que están definidos en la base de datos. Distintos objetos (fuzzyRules, fuzzyValues, products) pueden requerir la misma fuzzyVariable o fuzzyValue. Es por esto que usamos este contenedor global de objetos para ser accedido oportunamente por los objetos que lo requieran y así evitar múltiples accesos a la base de datos.

Los objetos de la clase Repository poseen un conjunto de fuzzyVariables y otro de fuzzyValues, los cuales se cargan en el método initialize() que es invocado al crearse una instancia. De esta manera se recuperan todos los fuzzyValues y fuzzyVariables que existen en la base de datos.

En el método de inicialización del coordinator se crea una instancia de esta clase y luego este objeto es enviado como parámetro al crear los productAdvisors

para luego ser usada en la creación de las fuzzyRules y products y obtener directamente de este objeto las fuzzyVariables y fuzzyValues sin tener que hacer lecturas redundantes a la base de datos.

Para obtener un fuzzyValue o una fuzzyVariable del objeto repository se utilizan los métodos getFuzzyValue() y getFuzzyVariable() especificando el identificador correspondiente. De esta manera logramos una mejor performance en el proceso inicial de recuperación de datos.

El código correspondiente a la clase Repository se encuentra en la pág. 122.

4.2.9- PageSection

Esta clase modela las distintas secciones destinadas a la publicidad de una página web.

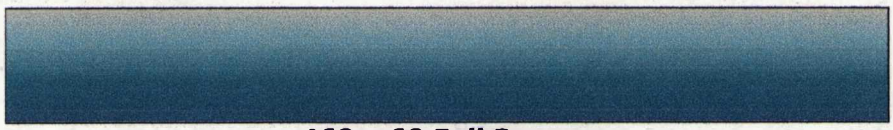
Los objetos de esta clase poseen un identificador, un nombre, el orden de prioridad de la sección con respecto a las demás secciones de la página y dos puntos que determinan su posición y tamaño. Estos puntos son el extremo izquierdo superior y el derecho inferior.

Los objetos de esta clase son accedidos por los objetos de la clase Product para poder mostrarse en la página.

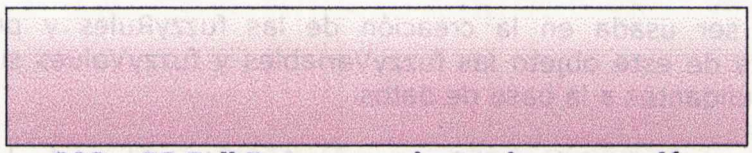
Para cada sitio web se definen la cantidad, forma, tamaño y ubicación en la página de las diferentes secciones, y luego los anuncios deben mostrarse en la sección indicada teniendo en cuenta el formato de la misma. Estos formatos fueron definidos por la Internet Advertising Bureau, IAB debido a la gran proliferación de banners de distintos tamaños y formatos que hubo cuando internet se convirtió en un medio comercial masivo (llegó a haber mas de 250 tamaños diferentes). Esta estandarización se llevó a cabo para hacer más fáciles las cosas a las agencias diseñadoras de banners y ayudar a desarrollar el negocio de la publicidad on-line.

Decidimos adoptar el banner ya que es la forma más atractiva y llamativa que toma la publicidad invitando al visitante a hacer click sobre éste, destacándose como herramienta visual y como acceso a más información ya que están relacionados a través de un link a una ubicación que detalla lo que menciona brevemente.

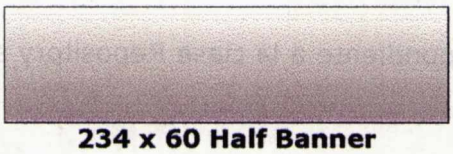
A continuación mostramos los banners con sus medidas específicas expresadas en pixeles.



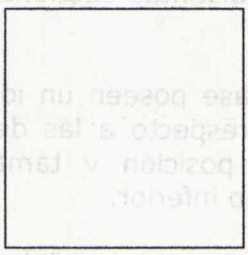
468 x 60 Full Banner



392 x 72 Full Banner con barra de navegación



234 x 60 Half Banner

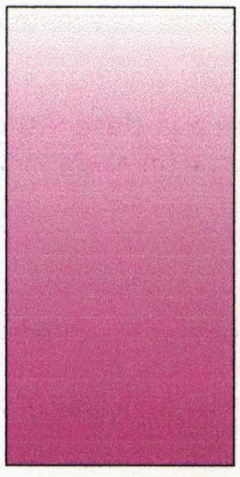


125 x 125 Ícono Cuadrado

120 x 90 Button 1

120 x 60 Button 2

88 x 31 Micro Button



120 x 240 Banner Vertical

El código correspondiente a la clase PageSection se encuentra en la pág. 123.

4.2.10- Visitor

Esta clase representa al visitante del sitio web. Entre los atributos de un visitante se encuentran: un identificador, un nombre, un apellido, una lista de productos y otra lista que contiene cada una de las características útiles para describir a un visitante del sitio.

La lista de características (visitorFeatures) contiene dos tipos de información del visitante:

- datos que el usuario inicializó al cargar su perfil en el sitio como podrían ser "edad", "ocupación", "estado civil", "sexo", "ingresos", "idioma", etc.
- datos del usuario obtenidos a partir de su comportamiento al navegar considerados de utilidad para la generación de reglas tales como "horarios frecuentes de navegación", "tiempo promedio de navegación", etc.

Por otro lado, la lista de productos contabiliza las veces que se mostró cada producto al visitante y la cantidad de veces que éste se interesó en la publicidad del mismo. Estos datos pueden ser de interés para quien desarrolle las reglas asociadas a un producto.

Se pretende que la base de conocimiento de los gustos y preferencias de los visitantes registrados aumente cada vez que éste visite el sitio. Se registran los productos que fueron seleccionados por nuestro sistema en cada oportunidad y las distintas elecciones de los visitantes. El visitante decide ignorar los anuncios o interesarse en ellos. Es importante destacar que medimos el interés del visitante

con el solo 'click' en el anuncio, mas allá de que luego este interés se concrete o no en una compra.

El código correspondiente a la clase Visitor se encuentra en la pág. 124.

4.2.11- Product

Esta clase sirve para modelar los diferentes productos que se ofrecen en las publicidades del sitio. Entre los datos de aquellos se encuentran: el identificador, la sección, una lista de imágenes, una lista de fuzzyVariables, y el repositorio de datos.

El identificador sirve para distinguir el producto unívocamente. La sección sirve para que el producto sepa en que lugar de la página debe mostrarse, en el caso de salir seleccionado para mostrar la publicidad. En la lista de imágenes guardamos el path de los archivos gráficos a mostrar. El repositorio de datos es usado en el período de inicialización para obtener las fuzzyVariables asociadas al producto, evitando el acceso a la base de datos.

El código correspondiente a la clase Product se encuentra en la pág. 127.

4.2.12- VisitorProduct

Los objetos de esta clase representan los productos vistos por un usuario particular del sitio.

VisitorProduct es subclase de la clase Producto y tiene dos datos adicionales que ayudan a evaluar la posibilidad de mostrar un determinado anuncio. Estos datos son: la cantidad de veces que se mostró el anuncio del producto al visitante y la cantidad de veces que éste se interesó en aquél. Esto es muy útil ya que algunos estudios indican que después de tres veces que una persona ve un mismo banner y no clickea en el ya no lo clickeará más. Otros opinan que el "banner burn out" (así se le llama a la sobre-exposición de un mismo banner) está en el rango de 200.000 impresiones y que habría que cambiarlo.

Estos datos estadísticos se mantienen con dos propósitos: el primero es la utilización de esta información en la generación de las fuzzyRules y el segundo es brindar al auspiciante cierta información estadística que puede consultarse on-line para observar la efectividad de la campaña, permitiendo auditar en tiempo real lo que sucede con sus anuncios. De este modo podría ofrecerse un sistema de estadísticas y acceso codificado a una página con los datos de los espacios contratados.

El código correspondiente a la clase VisitorProduct se encuentra en la pág. 129.

4.2.13- VisitorFeature

Representa cada una de las características que conforman el perfil de los visitantes. Su estructura posee un identificador, un nombre y un valor (el particular del usuario).

Mas allá de que las características que se deseen mantener son dinámicas y dependen del rubro del sitio, sugerimos algunas que permiten contar con los datos básicos para la generación de reglas y toma de decisiones: Edad, Ocupación, Nivel de Ingresos, Estado Civil, Sexo, hobbies, intereses generales, actividades físicas practicadas en forma regular, tipos de productos que compra, administración de compras, valor de compras, frecuencia de compras, tecnología usada, etc.

El código correspondiente a la clase VisitorFeature se encuentra en la pág. 130.

4.3- Muestra de la publicidad

El coordinador genera una cola de auspiciantes (ProductAdvisor) ordenada en forma decreciente de acuerdo a los resultados obtenidos por éstos mediante el "sistema de decisión de lógica difusa" descripto en pág. 24. Los resultados indican el posible interés para el visitante actual del sitio en el producto/servicio que oferta el auspiciante (ver pág. 37).

Sobre la cola ordenada aplicamos dos filtros:

Un filtro elimina aquellos auspiciantes para los cuales el grado de adecuidad sea inferior a un cierto valor preestablecido. Este valor es provisto por cada anunciante, lo que permite respetar de algún modo sus propios requerimientos y necesidades. Un grado de adecuidad bajo implica mayor flexibilidad en la selección del anuncio por lo que la cantidad de impresiones solicitadas por el anunciante se completará en un lapso de tiempo menor. Un grado de adecuidad alto produce el efecto contrario.

Ejemplo: Supongamos que tenemos en la cola 7 auspiciantes en el siguiente orden An5, An1, An3, An7, An4, An2 y An6. Los datos relevantes para realizar este primer filtro son: (An5, 0.9, 0.5), (An1, 0.83, 0.7), (An3, 0.69, 0.65), (An7, 0.64, 0.75), (An4, 0.56, 0.5), (An2, 0.48, 0.55), (An6, 0.32, 0.45), donde el primer componente de la terna es el nombre del auspiciante, el segundo componente es el grado de adecuidad obtenido y el tercer componente es el valor mínimo preestablecido por el auspiciante. El filtro elimina de la cola a An7, An2 y An6, por lo que la cola queda de la siguiente forma: An5, An1, An3 y An4.

Otro filtro elimina de la cola aquellos auspiciantes cuya publicidad haya sido mostrada al visitante actual más de un número predeterminado de veces sin que se interese en él. Cada auspiciante provee este valor que indica el número máximo de veces que desea mostrar una publicidad a un visitante.

Ejemplo: Supongamos tener la cola de auspiciantes An2, An1, An3, An5 y An4. Los datos relevantes para realizar este segundo filtrado son: (An2, 4, 4), (An1,

1, 3), (An3, 2, 3) , (An5, 0, 2), (An4, 2, 2) donde el primer componente de la terna es el nombre del auspiciante, el segundo componente es el número de veces que se ha mostrado la publicidad al visitante actual del sitio y el tercer componente es el número máximo de veces que se desea mostrar la publicidad a cada visitante si no se obtiene respuesta. El filtro elimina de la cola a An2 y An4 por lo que la cola queda de la siguiente forma: An1, An3 y An5.

Las secciones de la página destinadas a publicidad están clasificadas de acuerdo a la importancia de su ubicación y tamaño, esto determina su prioridad. Cada sección tiene un estado que indica si esta activa o no, lo cual permite deshabilitar temporalmente espacios de la página destinados a publicidad.

Para mostrar la publicidad se desencola el primer auspiciante y se le asigna la sección más prioritaria de la página, en la cual se mostrará su aviso. En el caso de que el auspiciante desee que su publicidad se muestre en determinadas secciones de la página y este no este dentro de ese grupo predeterminado se descarta el auspiciante y se toma el siguiente de la cola. Este proceso continúa para cada una de las secciones activas de la página.

Es necesario que el anunciante provea una versión de la publicidad para cada uno de los formatos de banner en los que desee publicitar. Se obtiene un mayor nivel de personalización si el anunciante proporciona distintas versiones de la misma publicidad adaptadas a las características del visitante que le parezcan más relevantes. Ejemplos: para la característica "idioma" presentar el anuncio en la propia lengua del visitante, para la característica "edad" proveer anuncios infantiles y para adultos.

Una vez seleccionadas las publicidades que se mostrarán en la página, se realiza el conteo de la cantidad de veces que se mostró cada publicidad en la fecha actual. Además, en el caso de que el visitante clickee sobre el anuncio se incrementa la cantidad de veces que despertó interés en él en esa fecha. Cuando se completan la cantidad de impresiones contratadas para ese auspiciante, éste deja de estar activo.

Esta información permite realizar cálculos estadísticos. Estos datos son mucho más importantes de lo que generalmente se piensa ya que aportan información estratégicamente valiosa para los anunciantes, quienes pueden seguir de cerca la repercusión de su campaña publicitaria.

Descripción del algoritmo de muestra de la publicidad:

Entrada: la cola de auspiciantes activos ordenada en forma decreciente según el grado de adecuidad de su publicidad al visitante actual del sitio, el visitante actual del sitio y las secciones activas de página.

Efecto: seleccionar y mostrar en la página actual tantas publicidades como secciones activas tenga ésta.

Para cada sección S_i activa y disponible de la página

Toma el primer auspiciante A_j de la cola

Mientras que no encuentre auspiciante adecuado

Si (grado de adecuidad calculado por A_j >= grado de adecuidad preestablecido en A_j) y (cant. de veces que se mostró la publicidad al visitante actual del sitio < cant. de veces preestablecido en A_j) y (S_i está entre las secciones aceptadas por A_j) entonces

Mostrar publicidad del auspiciante.

Incrementar veces que se mostró la publicidad al visitante.

Desencolar auspiciante.

Toma el siguiente auspiciante de la cola

donde, $S_{i\ i=1,\ldots,\text{cant secciones activas}}$

$A_{j\ j=1,\ldots,\text{cant auspiciantes}}$

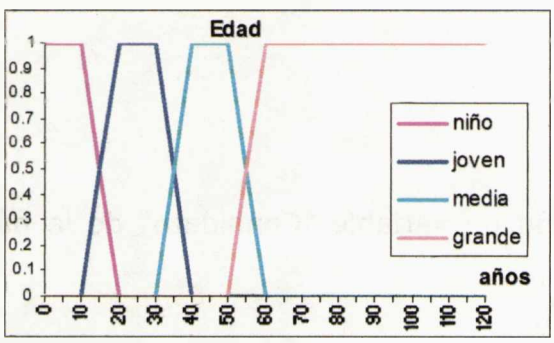
Capítulo 5

Ejemplo de trabajo

Supongamos que un sitio Web llamado "Tu Sitio" cuenta con los servicios de nuestro selector. En este sitio se han definido algunas fuzzyVariables que se considera que pueden ser de utilidad para los anunciantes. Tres de ellas, las fuzzyVariables "sexo", "edad" y "estado civil", son definidas de la siguiente forma:

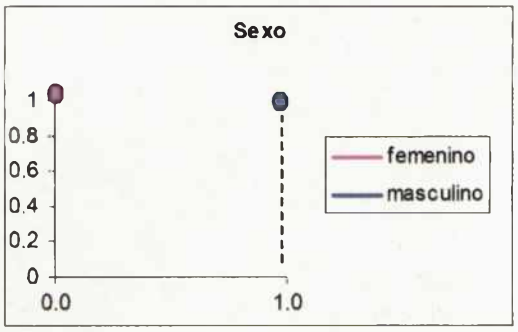
Nombre: "edad"
Unidad: años
Universo de discurso: El intervalo real [0,120]
Lista de fuzzySets: "niño", "joven", "media" y "grande".

Gráficamente:



Nombre: "sexo"
Unidad: -
Universo de discurso: el intervalo entero [0,1]
Lista de fuzzySets: "femenino" y "masculino".

Gráficamente:



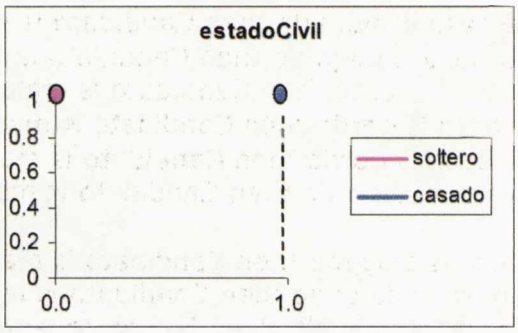
Nombre: "estadoCivil"

Unidad: -

Universo de discurso:

Lista de fuzzySets: "soltero", "casado"

Gráficamente:



Por último recordamos la definición de variable "Candidato" de la pág. 46. provista por el selector:

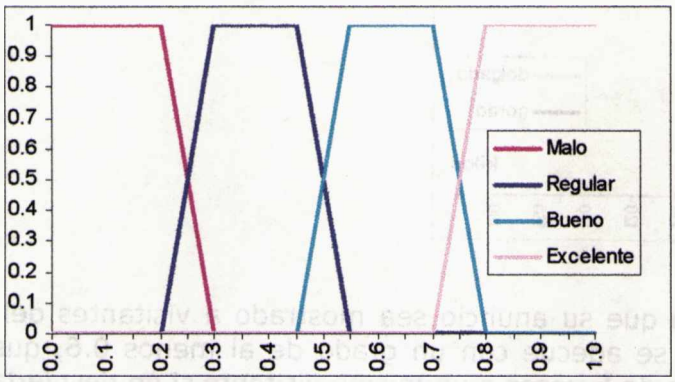
Nombre: "Candidato"
Unidad: frecuencia de clickeo
Universo de discurso: El intervalo real [0,1]
Lista de fuzzySets: "Excelente", "Bueno", "Regular" y "Malo".

La frecuencia de clickeo (FC) la definimos como:

$$FC = \frac{CAC}{CAM}$$

donde CAC es la cantidad de anuncios en los que el visitante clickeó y CAM es la cantidad total de anuncios mostrados al visitante.

Gráficamente:



Supongamos ahora que el anunciante de la empresa "ModasPepe" quiere publicitar en "Tu Sitio" su nueva línea de ropa de talles grandes para mujeres.

De acuerdo a su conocimiento del negocio, el anunciante define el siguiente Modelo Basado en Conocimiento Difuso:

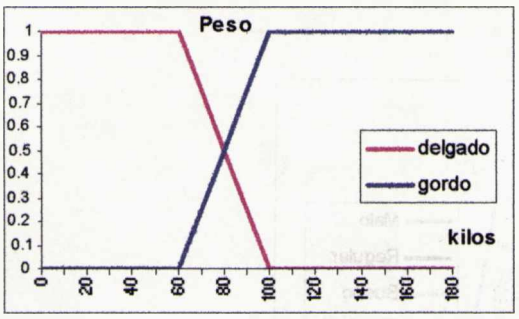
1)- If sexo is M and edad is niño and peso is delgado then Candidato is malo.
2)- If sexo is M and edad is joven and peso is delgado then Candidato is malo.
3)- If sexo is M and edad is media and peso is delgado then Candidato is malo.
4)- If sexo is M and edad is grande and peso is delgado then Candidato is malo.
5)- If sexo is M and edad is niño and peso is gordo then Candidato is malo.
6)- If sexo is M and edad is joven and peso is gordo then Candidato is malo.
7)- If sexo is M and edad is media and peso is gordo then Candidato is malo.
8)- If sexo is M and edad is grande and peso is gordo then Candidato is malo.

9)- If sexo is F and edad is niño and peso is delgado then Candidato is malo.
10)-If sexo is F and edad is joven and peso is delgado then Candidato is regular.
11)-If sexo is F and edad is media and peso is delgado then Candidato is regular.
12)-If sexo is F and edad is grande and peso is delgado then Candidato is bueno.
13)-If sexo is F and edad is niño and peso is gordo then Candidato is regular.
14)-If sexo is F and edad is joven and peso is gordo then Candidato is excelente.
15)-If sexo is F and edad is media and peso is gordo then Candidato is bueno.
16)-If sexo is F and edad is grande and peso is gordo then Candidato is bueno.

El anunciante decide usar las fuzzyVariables "edad" y "sexo" definidas por el sitio ya que responden a sus necesidades. A la fuzzyVariable "peso" como no se encuentra entre las predefinidas la define él mismo de la siguiente manera:

Nombre: "peso"
Unidad: kilos
Universo de discurso: el intervalo entero [0,180]
Lista de fuzzySets: "delgado" y "gordo".

Gráficamente:



Este anunciante desea que su anuncio sea mostrado a visitantes del sitio para los cuales su publicidad se adecue con un grado de al menos 0.6, que no se le muestre la publicidad más de 5 veces a un mismo visitante si no despierta interés y no tiene preferencias en cuanto a en que sección publicitaria de la página se muestre su aviso.

Supongamos también que "BancoRio" quiere promocionar un sistema de prestamos personales en "Tu Sitio".

De acuerdo a los requisitos que "BancoRio" impone, define las siguientes reglas:

1)- If estadoCivil is soltero and ingresos is bajo then Candidato is malo

2)- If estadoCivil is soltero and ingresos is medio then Candidato is bueno

3)- If estadoCivil is soltero and ingresos is alto then Candidato is excelente

4)- If estadoCivil is casado and ingresos is bajo then Candidato is malo

5)- If estadoCivil is casado and ingresos is medio then Candidato is malo

6)- If estadoCivil is casado and ingresos is alto then Candidato is bueno

Donde las fuzzyVariables "estadoCivil" es la predefinida por el sitio e "ingresos" es definida por "BancoRio" de la siguiente forma:

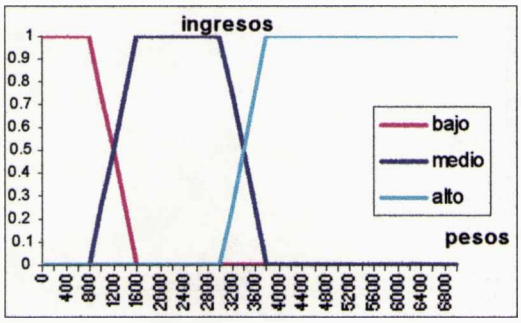
Nombre: "ingresos"

Unidad: pesos

Universo de discurso: El intervalo real [0,6000]

Lista de fuzzySets: "bajo", "medio"y "alto"

Gráficamente:



"BancoRio" quiere que su anuncio sea mostrado a visitantes del sitio para los cuales su publicidad se adecue con un grado de al menos 0.4, que no se le muestre la publicidad más de 3 veces a un mismo visitante si no despierta interés y desea que la publicidad se muestre en la sección principal de la página.

Supongamos que ingresa María Inés por primera vez al sitio. María Inés registra sus datos: 32 años de edad, 75 kilos de peso, casada e ingresos de 2800 pesos mensuales. Veamos como el selector evalúa si va a mostrarle las publicidades anteriores.

Cómo se evalúa la publicidad de "ModasPepe":

Inferencia: se obtiene el nivel de disparo de cada FuzzyRule y se aplica a la conclusión de la misma. El resultado es un FuzzySet.

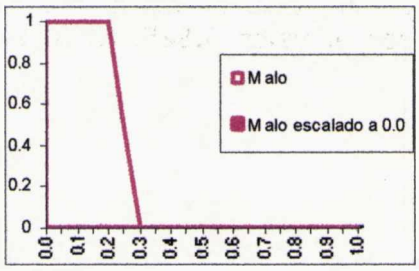
1)- If sexo is M and edad is niño and peso is delgado then Candidato is malo.

Se evalúa el FuzzySet "M" con el valor F y da 0.0.
Se evalúa el FuzzySet "niño" con el valor 32 y da 0.0.
Se evalúa el FuzzySet "delgado" con el valor 75 y da 0.625.

Obtenemos el nivel de disparo de la regla realizando:

min(0.0, 0.0, 0.625)=0.0

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "malo" por el valor 0.0. El fuzzySet que se obtiene es:



Para las reglas 2), 3), 4), 5), 6), 7) y 8) El resultado será idéntico a 1) porque en todas se escala el nivel de disparo 0.0 con el FuzzySet "malo".

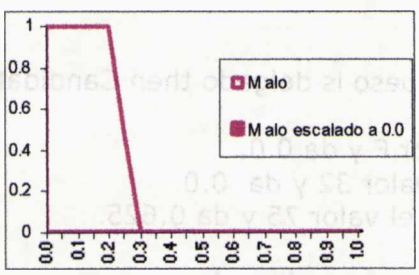
9)- If sexo is F and edad is niño and peso is delgado then Candidato is malo.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "niño" con el valor 32 y da 0.0.
Se evalúa el FuzzySet "delgado" con el valor 75 y da 0.625.

Obtenemos el nivel de disparo de la regla realizando:

$$\min(1.0, 0.0, 0.625)=0.0$$

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "malo" por el valor 0.0. El fuzzySet que se obtiene es:



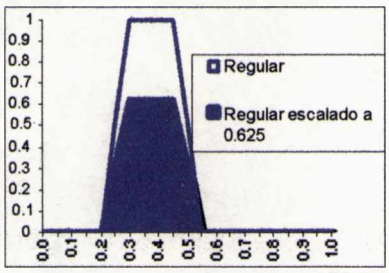
10)- If sexo is F and edad is joven and peso is delgado then Candidato is regular.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "joven" con el valor 32 y da 0.8.
Se evalúa el FuzzySet "delgado" con el valor 75 y da 0.625.

Obtenemos el nivel de disparo de la regla realizando:

$$\min(1.0, 0.8, 0.625)=0.625$$

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "regular" por el valor 0.625. El fuzzySet que se obtiene es:



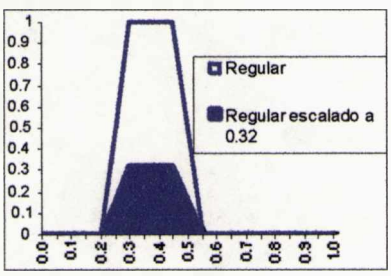
11)- If sexo is F and edad is media and peso is delgado then Candidato is regular.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "media" con el valor 32 y da 0.32.
Se evalúa el FuzzySet "delgado" con el valor 75 y da 0.875.

Obtenemos el nivel de disparo de la regla realizando:

$$\min(1.0, 0.32, 0.875)=0.32$$

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "regular" por el valor 0.625. El fuzzySet que se obtiene es:



12)- If sexo is F and edad is grande and peso is delgado then Candidato is bueno.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "grande" con el valor 32 y da 0.0.
Se evalúa el FuzzySet "delgado" con el valor 75 y da 0.875.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 0.0, 0.25)=0.0

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "bueno" por el valor 0.0. El fuzzySet que se obtiene es:

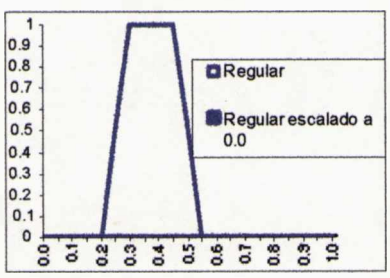
13)- If sexo is F and edad is niño and peso is gordo then Candidato is regular.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "niño" con el valor 32 y da 0.0.
Se evalúa el FuzzySet "gordo" con el valor 75 y da 1.0.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 0.9, 1.0)= 0.0

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "regular" por el valor 0.0. El fuzzySet que se obtiene es:



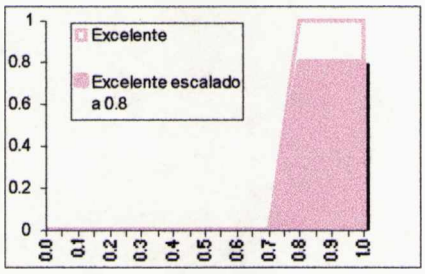
14)- If sexo is F and edad is joven and peso is gordo then Candidato is excelente.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "joven" con el valor 32 y da 0.8.
Se evalúa el FuzzySet "gordo" con el valor 75 y da 1.0.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 0.8, 1.0)= 0.8

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "excelente" por el valor 0.8. El fuzzySet que se obtiene es:



15)- If sexo is F and edad is media and peso is gordo then Candidato is bueno.

Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "media" con el valor 32 y da 0.2.
Se evalúa el FuzzySet "gordo" con el valor 75 y da 1.0.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 0.2, 1.0)= 0.2

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "bueno" por el valor 0.2. El fuzzySet que se obtiene es:

16)- If sexo is F and edad is grande and peso is gordo then Candidato is bueno.

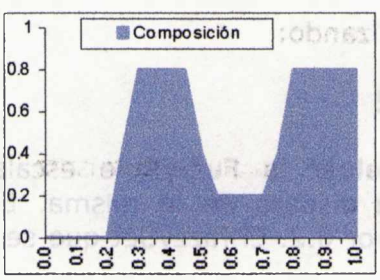
Se evalúa el FuzzySet "F" con el valor F y da 1.0.
Se evalúa el FuzzySet "grande" con el valor 32 y da 0.0.
Se evalúa el FuzzySet "gordo" con el valor 75 y da 1.0.

Obtenemos el nivel de disparo de la regla realizando:

$$\min(1.0, 0.0, 1.0) = 0.0$$

Obtenemos el FuzzySet resultante de evaluar la FuzzyRule escalando la conclusión de la FuzzyRule por el nivel de disparo de la misma. Es decir, multiplicamos el FuzzySet "bueno" por el valor 0.0. El fuzzySet que se obtiene es:

Composición: Se toma el punto máximo sobre todos los fuzzySets resultantes de la evaluación de cada una de las fuzzyRules:



Defusificación: La entrada para el proceso de defusificación es un FuzzySet (la salida de la etapa de composición) y la salida es un simple número concreto.

Recordemos:

$$Z^{*}_{CDA} = \frac{\sum_{j=1}^{n} z_j \, \mu_C(z_j)}{\sum_{j=1}^{n} \mu_C(z_j)}$$

Entonces:

$\sum_{j=1}^{n} z_j \, \mu_C(z_j)$ = 0.0*0.0 + 0.05*0.0 + 0.1*0.0 + 0.15*0.0 + 0.2*0.0 + 0.25*0. 3125 + 0.3*0. 625 + 0.35*0. 625 + 0.4*0. 625 + 0.45*0. 625 + 0.5*0.3125 + 0.55*0.2 + 0.6*0.2 + 0.65*0.2 + 0.7*0.2 + 0.75*0.4 + 0.8*0.8 + 0.85*0.8 + 0.90*0.8 + 0.95*0.8 + 1.0*0.8 = 5.571875.

y:

$\sum_{j=1}^{n} \mu_C(z_j)$ = 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.3125 + 0.625 + 0. 625 +0. 625 + 0.625 + 0.3125 + 0.2 + 0.2 + 0.2 + 0.2 + 0.4 + 0.8 + 0.8 + 0.8 + 0.8 + 0.8 = 8.325.

Por lo tanto el grado de en que se adecua María Inés a la publicidad de "ModasPepe" es 5.9/9.2 = 0.64130435.

Como el anunciante deseaba que su anuncio se mostrara a visitantes cuyo grado de adecuidad sea mayor a 0.6 y como la cantidad de veces que María Inés vio anteriormente la publicidad es menor a 5 (aún no la ha visto) se le muestra el aviso.

Cómo se evalúa la publicidad de "BancoRío":

Inferencia: se obtiene el nivel de disparo de cada FuzzyRule y se aplica a la conclusión de la misma.

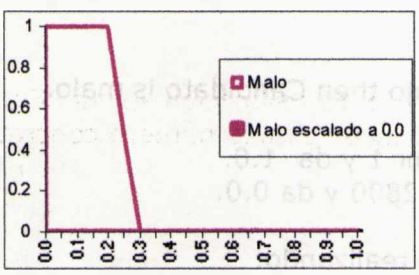
1)- If estadoCivil is soltero and ingresos is bajo then Candidato is malo.

Se evalúa el FuzzySet "soltero" con el valor 1 y da 0.0.
Se evalúa el FuzzySet "bajo" con el valor 2800 y da 0.0.

Obtenemos el nivel de disparo de la regla realizando:

min(0.0, 0.0)=0.0

Escalamos el nivel de disparo de la regla por la conclusión de la misma multiplicando el FuzzySet "malo" por el valor 0.0.



2)- If estadoCivil is soltero and ingresos is medio then Candidato is bueno.

Se evalúa el FuzzySet "soltero" con el valor 1 y da 0.0.
Se evalúa el FuzzySet "medio" con el valor 2800 y da 1.0.

Obtenemos el nivel de disparo de la regla realizando:

min(0.0, 1.0)=0.0

Escalamos el nivel de disparo de la regla por la conclusión de la misma multiplicando el FuzzySet "bueno" por el valor 0.0.

3)- If estadoCivil is soltero and ingresos is alto then Candidato is excelente.

Se evalúa el FuzzySet "soltero" con el valor 1 y da 0.0.
Se evalúa el FuzzySet "alto" con el valor 2800 y da 0.0.

Obtenemos el nivel de disparo de la regla realizando:

min(0.0, 0.0)=0.0

Escalamos el nivel de disparo de la regla por la conclusión de la misma multiplicando el FuzzySet "excelente" por el valor 0.0.

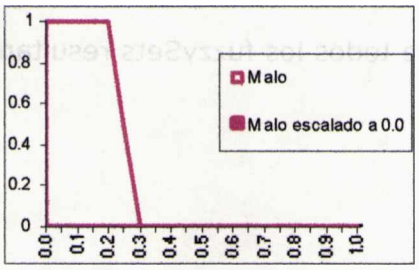
4)- If estadoCivil is casado and ingresos is bajo then Candidato is malo.

Se evalúa el FuzzySet "casado" con el valor 1 y da 1.0.
Se evalúa el FuzzySet "bajo" con el valor 2800 y da 0.0.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 0.0)=0.0

Escalamos el nivel de disparo de la regla por la conclusión de la misma multiplicando el FuzzySet "malo" por el valor 0.0.



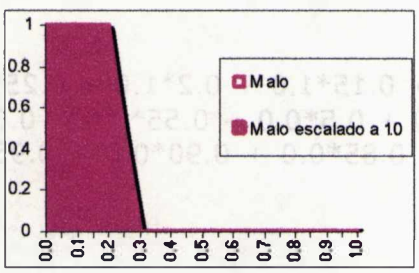
5)- If estadoCivil is casado and ingresos is medio then Candidato is malo.

Se evalúa el FuzzySet "casado" con el valor 1 y da 1.0.
Se evalúa el FuzzySet "medio" con el valor 2800 y da 1.0.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 1.0)=1.0

Escalamos el nivel de disparo de la regla por la conclusión de la misma multiplicando el FuzzySet "malo" por el valor 0.0.



6)- If estadoCivil is casado and ingresos is alto then Candidato is bueno.

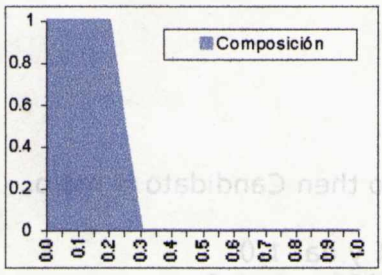
Se evalúa el FuzzySet "casado" con el valor 1 y da 1.0.
Se evalúa el FuzzySet "alto" con el valor 2800 y da 0.0.

Obtenemos el nivel de disparo de la regla realizando:

min(1.0, 0.0)=0.0

Escalamos el nivel de disparo de la regla por la conclusión de la misma multiplicando el FuzzySet "bueno" por el valor 0.0.

Composición: Se toma el punto máximo sobre todos los fuzzySets resultantes de la evaluación de cada una de las fuzzyRules.



Defusificación: La entrada para el proceso de defusificación es un FuzzySet (la salida de la etapa de composición) y la salida es un simple número concreto.

Sabemos que:

$$Z^{*}_{CDA} = \frac{\sum_{j=1}^{n} z_j \, \mu_C(z_j)}{\sum_{j=1}^{n} \mu_C(z_j)}$$

Entonces:

$\sum_{j=1}^{n} z_j \, \mu_C(z_j)$ = 0.0*1.0 + 0.05*1.0 + 0.1*1.0 + 0.15*1.0 + 0.2*1.0 + 0.25*0. 5 + 0.3*0. 0 + 0.35*0. 0 + 0.4*0. 0 + 0.45*0. 0 + 0.5*0.0 + 0.55*0.0 + 0.6*0.0 + 0.65*0.0 + 0.7*0.0 + 0.75*0.0 + 0.8*0.0 + 0.85*0.0 + 0.90*0.0 + 0.95*0.0 + 1.0*0.0 = 0.625

y:

$\sum_{j=1}^{n} \mu_C(z_j)$ = 1.0 + 1.0 + 1.0 + 1.0 + 1.0 + 0.5 + 0.0 + 0.0 +0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 = 5.5

Por lo tanto el grado de en que se adecua María Inés a la publicidad de "BancoRio" es 0.625/5.5 = 0.113637

Como el anunciante deseaba que su anuncio se mostrara a visitantes cuyo grado de adecuidad sea mayor a 0.4 a María Inés no se le muestra esta publicidad.

Capítulo 6

Notas finales

6.1- Conclusiones

En el presente trabajo diseñamos un selector de publicidades para ser incorporado a cualquier sitio web.

Para realizar este selector fue necesario llevar a cabo un estudio profundo sobre lógica difusa ya que se trataba de un tema desconocido para nosotras. La lógica difusa nos resultó sumamente interesante debido a su forma de tratar conceptos que no tienen una definición clara.

La elección de lógica difusa se mostró muy conveniente porque le permite al anunciante representar el conocimiento no estructurado que posee de su negocio en forma explícita mediante reglas del tipo IF-THEN .

Otra elección de suma importancia fue la de permitir al anunciante definir sus propios fuzzySets y fuzzyVariables en vez de estar sujeto a un número de fuzzySets y fuzzyVariables estáticos y arbitrarios definidos en el selector. En un principio contemplamos la posibilidad de manejarnos con un conjunto predeterminado de los mismos pero luego llegamos a la conclusión de que sería demasiado rígido. Esta elección le ofrece al anunciante la posibilidad de definir lo que realmente necesita y en caso de que luego requiera algunos ajustes fácilmente puede mejorar, adaptar y modificar lo que usa.

Una de las ventajas de este selector es que si los anunciantes cambian las reglas no hay que recompilar el código, pues éstas se cargan en tiempo de ejecución. Con otros diseños el cambio de comportamiento o ajuste resulta mucho mas laborioso, llegando incluso a requerir recodificar gran parte del sistema.

Otra de las ventajas es que el proceso de selección de las publicidades se realiza a partir de los datos ingresados por el visitante. Esto nos permite ajustar los anuncios con base en lo que a los clientes realmente les interesa en lugar de hacerlo basándonos en lo que creemos que quieren.

Por lo tanto consideramos que hemos logrado el objetivo de realizar un diseño que permita ayudar a los anunciantes a tomar decisiones en cuanto a los visitantes de sitios web con el objetivo de que la publicidad resulte beneficiosa para el anunciante y a la vez sea de utilidad para el visitante.

6.2- Perspectivas

6.2.1- Extensión de la funcionalidad

El selector que hemos definido puede ser mejorado mediante la inclusión de otros mecanismos existentes en lógica difusa. Sería interesante proveer otros métodos de inferencia (recordar que mencionamos que existen alrededor de 40 funciones de implicación difusa distintas descriptas en la literatura existente); otros métodos para realizar la composición de las reglas como podría ser el método de la suma por mencionar uno de los más conocidos; del mismo modo se podrían proporcionar otros métodos de defusificación.

Otra posible mejora sería la de aplicar más de un modificador a los fuzzySets como por ejemplo para lograr expresiones del tipo "muy ligeramente debajo de".

6.2.2- Aplicaciones

Este sistema podría ser propuesto en un sitio de una determinada empresa donde al conocimiento proporcionado en forma explícita por el visitante y al conocimiento obtenido a partir de su comportamiento al navegar se incorpora el conocimiento concreto de compras que se conoce de ese visitante.

Apéndice

Documentación técnica de la aplicación

Clase Coordinator

Definición de la clase:

```
import java.util.Vector;
import ar.com.app.types.*;
import ar.com.app.connections.*;
import ar.com.app.system.*;
import java.awt.Point;
public class Coordinator extends Object {
        public Vector productAdvisors;
        public Vector pageSections;
        public Repository repository;
}
```

Métodos públicos:

```
/**
 * Método constructor del objeto coordinador.
 */
public Coordinator() {
        super();
        initialize();
}


/**
 * Agrega un productAdvisor a la lista.
 */
public void addProductAdvisor(ProductAdvisor productAdvisor) {
        this.productAdvisors.addElement(productAdvisor);
        }


/**
 * Agrega un sección a la lista.
 */
public void addSection(PageSection section) {
        pageSections.addElement(section);
        }


/**
 * Retorna el productAdvisor de la posición i.
 */
public ProductAdvisor advisorsAt(int i) {
        return (ProductAdvisor) productAdvisors.elementAt(i);
        }

/**
 * Obtiene los datos del usuario y la información de los productos ofrecidos y accedidos históricamente.
 */
public Visitor getVisitor(int userId) {
```

```
        return new Visitor(userId);
        }


/**
 * Se cargan las secciones activas y los productAdvisors activos.
 */
public void initialize() {
        String sCmd;

                ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+
                Constants.ODBC_FUZZY_DSN, Constants.ODBC_FUZZY_USER,
                Constants.ODBC_FUZZY_PASSWORD);

        this.repository =  new Repository();
        sCmd= "active_Sections" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        this.pageSections = new Vector();
        con.first();
        while (!con.isEof()) {
                addSection(new PageSection(con.getField("idSection").toInt()));
                con.next();
                }

        sCmd= "active_ProductAdvisors" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        this.productAdvisors = new Vector();
        con.first();
        while (!con.isEof()) {
                addProductAdvisor(new ProductAdvisor(con.getField("idProductAdvisor").toInt(), this.repository));
                con.next();
                }
        }


/**
 *Se invoca cuando un usuario visita la página.
 */
public boolean login(int userId) {
        Visitor visitor;

        visitor = this.getVisitor (userId);
        if (visitor == null )
                return false;
        start(visitor);
        return true;
        }

        /**
         * Retorna la sección de la posición i.
         */
        public PageSection pageSectionAt(int i) {
                return (PageSection) pageSections.elementAt(i);
                }


        /**
         * Retorna el productAdvisor de la posición i.
         */
        public ProductAdvisor productAdvisorsAt(int i) {

                return (ProductAdvisor) productAdvisors.elementAt(i);

                }

        /**
         * Elimina el productAdvisor de la posición i.
```

```
        */
        public void removeProductAdvisor(Integer pos) {
                this.productAdvisors.removeElement(pos);
                }


/**
 * Comienza la selección de anuncios para el usuario logueado.
 * Evalúa todos los productAdvisors y los ingresa a una cola ordenada.
 * Según la cantidad de secciones activas le avisa a algunos productAdvisors que muestren su producto.
 */
public void start(Visitor visitor) {
        PriorityQueue queue;
        int i;

        queue = new PriorityQueue();

        for (i=0; i< this.productAdvisors.size(); i++){
                productAdvisorsAt(i). setData (visitor,queue);
                productAdvisorsAt(i).start();
        }

        while(queue.size() != this.productAdvisors.size()) {
                // espero a que esten todos los resultados
        }

        for (i=0; i< this.pageSections.size(); i++)
                {
                        if (queue.size() > 0)
                        queue.dequeue().showAd(pageSectionAt(i),visitor);
                }

        for (i=0; i< this.productAdvisors.size(); i++){
                productAdvisorsAt(i).destroy();
        }
}
```

Clase Product Advisor

Definición de la clase:

```
import java.util.Vector;
 import ar.com.app.types.*;
 import ar.com.app.connections.*;
public class ProductAdvisor extends Thread {
        public int idAdvisor;
        public String advisor;
        public Product product;
        public Vector rules;
        public double result; /* resultado de la evaluacion de todas las reglas*/
        public Visitor visitor;
        public PriorityQueue queue;
        public Repository repository;

        public double minDegree;
        public double maxAdPerVisitor;
        public double adImpressions;
        public Vector aceptedSecctions;
}
```

Métodos públicos:

```
/**
 * Agrega la regla dada a la lista de reglas del producto.
 */
```

```
public void addRule(FuzzyRule rule) {
        this.rules.addElement(rule);
}


/**
 * Retorna el producto asociado al productAdvisor.
 */
public Product getProduct() {
        return this.product         ;
        }


/**
 * Carga los datos del ProductAdvisor.
 */
public void initialize() {
        String sCmd;

                ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+
                Constants.ODBC_FUZZY_DSN, Constants.ODBC_FUZZY_USER,
                Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "select_ProductAdvisor " + "'" + this.idAdvisor + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        con.first();
        while (!con.isEof()) {
                this.product = new Product (con.getField("idProduct").toInt(), this.repository);
                this.advisor= con.getField("name").toString();
                con.next();
                }

        sCmd= "advisor_Rules " + "'" + this.idAdvisor + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        this.rules = new Vector();
        con.first();
        while (!con.isEof()) {
                addRule(new FuzzyRule(con.getField("idRule").toInt(), this.repository));
                con.next();
                }
        }


/**
        *Retorna true si el resultado obtenido del sistema de decisión de  lógica difusa
        *para el  auspiciante receptor del mensaje es menor o igual al del productAdvisor
        *pasado como parámetro.
 */
public boolean isLower(ProductAdvisor productAdvisor) {
        return this.result <= productAdvisor.result;
        }


/**
 * Elimina la regla dada a las reglas del producto.
 */
public void removeRule(FuzzyRule rule) {
        this.rules.removeElement(rule);
}


/**
 * Retorna la regla de la posición i
 */
public FuzzyRule ruleAt(int i) {
        return (FuzzyRule) this.rules.elementAt(i);
        }
```

```
/**
 * Retorna las reglas del producto.
 */
public Vector rules() {
	return this.rules;
}


/**
 * Setea las reglas del producto.
 */
public void rules(Vector rules) {
	this.rules = rules;
}


/**
 * Retorna la cantidad de regias.
 */
public int rulesCount () {
	return this.rules.size();
	}



/**
 * 1 - Evaluación de la regla
	a - Calculo el nivel de disparo de cada regla con el perfil dado.
					NDi = Min [Aij(Xj*)]	j= 1..p
	b - Calculo la salida efectiva de la regla pesando su consecuente por su nivel de disparo
					Fi(y) = ND * Bi(y)

   2 - Obtengo la salida difusa efectiva donde: F(y)= Max [Fi(y)]  j= 1..n


   3 - Defusificar
 */
public void run() {
	int i;
	FuzzySet fSet;
	FuzzySet r;

	fSet = FuzzySet.neutral();
	for ( i = 0;i < rulesCount(); i++)
	{
		r = ruleAt(i).evaluate(visitor);
		fSet = fSet.fuzzyUnion(r);
	}

	this.result = fSet.defussificate();
	queue.enqueue(this);

	}


/**
 * Asocia un visitante y una cola al productAdvisor.
 */
public void setData(Visitor visitor, PriorityQueue queue) {
	this.queue = queue;
	this.visitor = visitor;
	}


/**
 * Setea el producto al productAdvisor.
 */
public Product setProduct(Product aProduct) {
	this.product = aProduct;
	return aProduct   ;
	}
```

```
/**
 * Muestra el aviso que se ajuste a la sección dada de la página.
 */
public void showAd(PageSection section, Visitor visitor) {
        product.showAd(section, visitor);
}
```

Clase FuzzyRule

Definición de la clase:

```
import java.util.Vector;
import ar.com.app.types.*;
import ar.com.app.connections.*;
import ar.com.app.system.*;
public class FuzzyRule extends Object{
        public int idFuzzyRule;

        public FuzzyValueVector rulePreconditions; /*son FuzzyValues*/
        public FuzzyValue ruleConsequent;
        public Vector ruleInputValues;
        public Repository repository;
}
```

Métodos públicos:

```
/**
        * Método constructor de la fuzzyRule.
        */
public FuzzyRule(int idRule, Repository repository) {
                super();
                this.idFuzzyRule = idRule;
                this.repository =repository;
                initialize();
                }

/**
 * Agrega una precondición a la fuzzyRule.
 */
public void addPrecondition(FuzzyValue precondition) {
        this.rulePreconditions.addElement(precondition);
        }

/**
 * Retorna el fuzzySet resultante de la evaluación de la fuzzyRule.
 */
public FuzzySet evaluate (Visitor visitor) {
                loadInputValues(visitor);
                return this.getConsequent().weightOf(this.fireLevel());
        }


/**
  * Descripción de los pasos:
                1- Evaluar cada una de las precondiciones utilizando el perfil del usuario.
                2- Calcular el mínimo numérico entre los resultados del paso anterior.
                3- Retornar  ese mínimo (es el nivel de disparo).
  */
public double fireLevel() {
                int i;
                double min, temp;
```

```
                min= 1;
                for ( i = 0;i < this.rulePreconditions.size(); i++){
                        temp = rulePreconditionAt(i).evaluate(ruleInputValueAt(i));
                        if (temp < min){
                                min = temp;
                        }
                }
                return min;
        }


/**
 * Retorna el consecuente de la fuzzyRule.
 */
public FuzzyValue getConsequent() {
                return this.ruleConsequent;
}


/**
 * Inicializa la fuzzyRule.
 */
public void initialize() {
                String sCmd;

                ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+
                Constants.ODBC_FUZZY_DSN, Constants.ODBC_FUZZY_USER,
                Constants.ODBC_FUZZY_PASSWORD);

                this.ruleInputValues = new Vector();
                sCmd= "rule_preconditions " + "'" + this.idFuzzyRule + "'" ;
                if (!con.executeQuery(sCmd)){
                        con.close();
                }
                this.rulePreconditions = new FuzzyValueVector();
                con.first();
                while (!con.isEof()) {
                    addPrecondition(this.repository.getFuzzyValue(con.getField("idFuzzyValue").toInt()));
                   con.next();
                 }

                sCmd= "select_rule_consequent " + "'" + this.idFuzzyRule + "'" ;
                if (!con.executeQuery(sCmd)){
                        con.close();
                }
                con.first();
                while (!con.isEof()) {
                  this.ruleConsequent=this.repository.getFuzzyValue(con.getField("idfuzzyValue").toInt());
                  con.next();
                }
    }


/**
 * Carga los datos correspondientes al perfil del visitante.
 */
public void loadInputValues(Visitor visitor) {
        int i;
        for ( i = 0;i < this.rulePreconditions.size(); i++)

        this.ruleInputValues.addElement(visitor.getFeature(rulePreconditionAt(i).fuzzyVariable.feature));
        }


/**
 * Elimina una precondición a la fuzzyRule.
 */
public void removePrecondition(FuzzyValue precondition) {
        this.rulePreconditions.removeElement(precondition);
```

```
}

/**
 * Retorna el valor de entrada de la fuzzyRule de la posición i.
 */
public double ruleInputValueAt(int i) {
        return ((VisitorFeature)this.ruleInputValues.elementAt(i)).value;
}

/**
 * Retorna la precondición de la posición i.
 */
public FuzzyValue rulePreconditionAt(int i) {
        return (FuzzyValue) this.rulePreconditions.elementAt(i);
}

/**
 * Setea el consecuente de la fuzzyRule.
 */
public void setConsequent(FuzzyValue fuzzyValue) {
        this.ruleConsequent = fuzzyValue;
}
```

Clase FuzzyValue

Definición de la clase:

```
import ar.com.app.types.Constants;
import ar.com.app.connections.ConnectionSQLAgent;
public class FuzzyValue extends Object {
        public int idFuzzyValue;

        public FuzzySet fuzzySet;
        public FuzzyModifier modifier;
        public FuzzyVariable fuzzyVariable;

}
```

Métodos Públicos:

```
/**
 * Construye un nuevo FuzzyValue el cual tiene a la FuzzyVariable como argumento y crea un FuzzySets
 de los dos arreglos de valores double.
 */
public FuzzyValue(FuzzyVariable fuzzyVariable, double[] x, double[] y, int numPoints){
        super();
        this.fuzzyVariable = fuzzyVariable;
        this.fuzzySet= new FuzzySet("",x,y, numPoints);
}

/**
 * Setea los datos del fuzzyValue.
 */
public FuzzyValue(FuzzyVariable fuzzyVariable, FuzzySet fuzzySet) {
        super();
        this.fuzzyVariable = fuzzyVariable;
        this.fuzzySet= fuzzySet;
        }

/**
 * Construye un nuevo FuzzyValue el cual tiene a la FuzzyVariable como argumento y crea un FuzzySets
```

```
 de los dos arreglos de valores double.
 */
public FuzzyValue(FuzzyVariable fuzzyVariable, SetPoint[] setPoints,int numPoints){
        super();
        this.fuzzyVariable = fuzzyVariable;
        this.fuzzySet = new FuzzySet ("", setPoints, numPoints);
        }

/**
 * Método constructor del fuzzyValue.
 */
public FuzzyValue(int idFuzzyValue){
        super();
        this.idFuzzyValue = idFuzzyValue;
        initialize();
        }

/**
 * Aplica el modificador al fuzzyValue.
 */
public void applyModifier() {
        this.fuzzySet = this.modifier.call(this.fuzzySet);
        }

/**
 * Retorna true si el fuzzyValue es igual al que se pasa como parámetro,
 * false en caso contrario.
 */
        public boolean equals(FuzzyValue fuzzyValue ) {
        return (this.idFuzzyValue == fuzzyValue.idFuzzyValue);
        }


/**
 * Retorna true si el fuzzyValue es igual al que se pasa como parámetro,
 * false en caso contrario.
 */
        public boolean equals(Object fuzzyValue ) {
        return equals((FuzzyValue) fuzzyValue);
        }

/**
 * Retorna el resultado de evaluar con x el fuzzySet asociado.
 */
public double evaluate(double x) {
        return fuzzySet.evaluate(x);
        }

/**
 * Inicializa el fuzzyValue.
 */
public void initialize() {
        String sCmd;
        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "select_fuzzyValue " + "'" + this.idFuzzyValue + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        con.first();
        if (!con.isEof()) {
                this.fuzzyVariable = new FuzzyVariable(con.getField("idfuzzyVariable").toInt());
                this.modifier = FuzzyModifier.createFuzzyModifier(con.getField("idModifier").toInt());
                setFuzzySet(con.getField("idFuzzySet").toInt(),con.getField("idShape").toInt());
                applyModifier();
                }
        }


/**
 * Crea el fuzzySey correspondiente a la forma dada.
```

```
 */
public void setFuzzySet( int idFuzzySet,int idShape) {

if (idShape == 1)
                this.fuzzySet= (new FuzzySet(idFuzzySet));
        else if (idShape == 2)
                this.fuzzySet= (new LFuzzySet(idFuzzySet));
        else if (idShape == 3)
                this.fuzzySet= (new LeftLinearFuzzySet(idFuzzySet));
        else if (idShape == 4)
                this.fuzzySet= (new SFuzzySet(idFuzzySet));
        else if (idShape == 5)
                this.fuzzySet= (new LRFuzzySet(idFuzzySet));
        else if (idShape == 6)
                this.fuzzySet= (new PIFuzzySet(idFuzzySet));
        else if (idShape == 7)
                this.fuzzySet= (new TrapezoidFuzzySet(idFuzzySet));
        else if (idShape == 8)
                this.fuzzySet= (new RectangleFuzzySet(idFuzzySet));
        else if (idShape == 9)
                this.fuzzySet= (new TriangleFuzzySet(idFuzzySet));
        else if (idShape == 10)
                this.fuzzySet= (new SingletonFuzzySet(idFuzzySet));
        else if (idShape == 11)
                this.fuzzySet= (new RFuzzySet(idFuzzySet));
        else if (idShape == 12)
                this.fuzzySet= (new RightLinearFuzzySet(idFuzzySet));
        else if (idShape == 13)
                this.fuzzySet= (new ZFuzzySet(idFuzzySet));
        }


/**
 * Retorna el fuzzySet resultante de escalar el fuzzySet asociado por el fireLevel dado.
 */

public FuzzySet weightOf(double fireLevel) {
        return this.fuzzySet.fuzzyScale(fireLevel);
        }
```

Clase FuzzyVariable

Definición de la clase:

```
import java.util.Vector;
import ar.com.app.types.Constants;
import ar.com.app.connections.ConnectionSQLAgent;
public class FuzzyVariable extends Object {
        public int idFuzzyVariable;

        public java.lang.String name;
        public String units;
        public double lower;
        public double upper;
        public Vector fuzzySets;
        public VisitorFeature feature;
```

Métodos públicos:

```
/**
 * Método constructor de la fuzzyVariable.
 */
public FuzzyVariable(int idFuzzyVariable, java.lang.String name,double UODlower,
                                        double UODupper,java.lang.String u)
```

```
                                {
        super();
        this.idFuzzyVariable = idFuzzyVariable;
        this.name = name;
        this.units = u;
        this.lower= UODlower;
        this.upper=UODupper;
        initialize();
}


/**
 * Agrega el FuzzySet a la lista de fuzySets de la FuzzyVariable.
 */
void addFuzzySet( int idFuzzySet,int idShape) {

        if (idShape == 1)
                this.fuzzySets.addElement (new FuzzySet(idFuzzySet));
        else if (idShape == 2)
                this.fuzzySets.addElement (new LFuzzySet(idFuzzySet));
        else if (idShape == 3)
                this.fuzzySets.addElement (new LeftLinearFuzzySet(idFuzzySet));
        else if (idShape == 4)
                this.fuzzySets.addElement (new SFuzzySet(idFuzzySet));
        else if (idShape == 5)
                this.fuzzySets.addElement (new LRFuzzySet(idFuzzySet));
        else if (idShape == 6)
                this.fuzzySets.addElement (new PIFuzzySet(idFuzzySet));
        else if (idShape == 7)
                this.fuzzySets.addElement (new TrapezoidFuzzySet(idFuzzySet));
        else if (idShape == 8)
                this.fuzzySets.addElement (new RectangleFuzzySet(idFuzzySet));
        else if (idShape == 9)
                this.fuzzySets.addElement (new TriangleFuzzySet(idFuzzySet));
        else if (idShape == 10)
                this.fuzzySets.addElement (new SingletonFuzzySet(idFuzzySet));
        else if (idShape == 11)
                this.fuzzySets.addElement (new RFuzzySet(idFuzzySet));
        else if (idShape == 12)
                this.fuzzySets.addElement (new RightLinearFuzzySet(idFuzzySet));
        else if (idShape == 13)
                this.fuzzySets.addElement (new ZFuzzySet(idFuzzySet));
        }


/**
 * Retorna true si la fuzzyVariable recibida como parámetro es igual
 *  a la fuzzyVariable que recibe el mensaje, false en caso contrario.
 */
        public boolean equals(FuzzyVariable fuzzyVariable ) {
        return (this.idFuzzyVariable == fuzzyVariable.idFuzzyVariable);
        }


/**
 * Retorna true si la fuzzyVariable recibida como parámetro es igual
  a la fuzzyVariable que recibe el mensaje, false en caso contrario.
 */
        public boolean equals(Object fuzzyVariable ) {
        return equals((FuzzyVariable) fuzzyVariable);
        }


/**
 * /**
 * Retorna true si el identificador recibido como parámetro es igual
  al de la fuzzyVariable que recibe el mensaje, false en caso contrario.
 */
        public boolean equalss(Integer idFuzzyVariable ) {
        return (this.idFuzzyVariable == idFuzzyVariable.intValue());
        }
```

```
/**
 * Retorna la característica del visitante que corresponde al concepto
 * que representa la fuzzyVariable.
 */
public VisitorFeature feature(Visitor visitor) {
        return visitor.getFeature(this.feature);
}


/**
 * Retorna el nombre de la fuzzyVariable.
 */
public String getName() {
        return this.name;
}


/**
 * Inicializa la fuzzyVariable.
 */
public void initialize() {
        String sCmd;

        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "select_fuzzyVariable " + "'" + this.idFuzzyVariable + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        con.first();
        while (!con.isEof()) {
                this.name= con.getField("name").toString();
                this.units= con.getField("units").toString();
                this.lower= con.getField("lower").toDouble();
                this.upper= con.getField("upper").toDouble();
                        this.feature = new
        VisitorFeature(con.getField("idFeature").toInt(),con.getField("featureName").toString());
                con.next();
                }

        sCmd= "select_fuzzyVariableSets " + "'" + this.idFuzzyVariable + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        fuzzySets = new Vector();
        con.first();
        while (!con.isEof()) {
                addFuzzySet(con.getField("idFuzzySet").toInt(), con.getField("idShape").toInt() ) ;
                con.next();
                }

}


/**
 * Setea el nombre de la fuzzyVariable.
 */
public void name(java.lang.String name) {
        this.name = name;
}
```

Clase FuzzySet

Definición de la clase:

```
import ar.com.app.types.Constants;
import ar.com.app.connections.ConnectionSQLAgent;
public  class FuzzySet extends Object {
        public int idFuzzySet;
        public java.lang.String name;
        protected int numPoints;
        protected ar.com.app.system.SetPoint[] set;
        protected static final int MAX_INITIAL_POINTS = 130;
}
```

Métodos públicos:

```
/**
 * Método constructor de FuzzySet.
 */
public FuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Método contructor de un fuzzySet.
 */
public FuzzySet(String name,SetPoint[] setPoints,int numPoints){
        super();
        this.name = name;
        this.numPoints = numPoints;
        this.set = setPoints;
}


/**
 * Retorna un singleton.
 */
public static FuzzySet neutral() {
        return new SingletonFuzzySet(0.0);
}


/**
 * Agrega el setPoint  en al final del arreglo.
 * El nuevo punto es agregado en el FuzzySet de modo de mantener los x en orden incremental consistente.
 *Si el valor de pertenencia es menor que 0 le da el valor 0.
 */
public void appendFuzzySet(FuzzySet fuzzySet) {
        int i;
        for(i=0;(i<fuzzySet.numPoints);i++)
                this.appendSetPoint(fuzzySet.getPoint(i));
 }


/**
 *Agrega el setPoint  en el fin del arreglo. El nuevo punto es agregado en el FuzzySet de modo
 *de mantener los x en orden incremental consistente.  Si el valor de pertenencia es menor
 *que 0 le da el valor 0.
 */
public void appendSetPoint(SetPoint point) {
        int i;
        int j;
```

```
        i=this.size()-1;
        while (i >= 0 & set[i].getX() > point.getX()) {
                set[i+1] = set[i];
                i--;
        }

        if (point.getY() >= 0.0){
         this.set[i+1] = point;
        }
        else {
         this.set[i+1] = new SetPoint(point.getX(),0);
        }
        this.numPoints=this.numPoints + 1;
}


/*
 * Calcula el factor de escala necesario para que el maximo valor y del fuzzy set sea 1.
 *Retorna 1/(maximo Y del fuzzySet).
 */
public double calculateScaleFactor() {
        return 1 / this.getMaxY();
}


/**
 * Chequea para asegurar que todos los valores x están en orden ascendente.
 *Retorna -1 si es así, caso contrario retorna el de la posición en que se encuentra el valor fuera de orden.
 */
public int checkXValueOrder() {
  int i;

  i= 0;
  while ((i< (this.size()-1)) & (set[i].getX() < set[i+1].getX())){
          i++;
          }
  if (i< (this.size()-1)){
        return -1;
  }
  else {return i;
          }
}


/**
 *Chequea para asegurar que todos los valores y están en  el intervalo [0,1].
 *Retorna -1 si es así, caso contrario retorna el de la posición  en que se encuentra el valor fuera de rango.
 */
public int checkYValueRange() {
  int i;

  i= 0;
  while ((i<=(this.size()-1)) & (set[i].getY() >=0) & (set[i].getY() <=1)){
          i++;
          }
  if (i == this.size()){
        return -1;
        }
  else {
          return i;
        }
}


/**
*Retorna un fuzzySet que es una copia del FuzzySet.
*/
public FuzzySet copy(){
        FuzzySet newFuzzySet;
        int i;

        newFuzzySet = new FuzzySet();
```

```
        newFuzzySet.name = this.name;
        newFuzzySet.numPoints = this.size();
        for (i=0;(i< this.size()); i++)
          newFuzzySet.set[i] = new SetPoint(this.set[i]);
        return newFuzzySet;
        }


/**
 * Retorna el resultado de la defusificación del FuzzySet.
 */
 double defussificate() {
        int i;
        double x;
        double sumUp;
        double sumDown;

        sumUp=0;
        sumDown=0;
        x=0;
        for (i=0;(i< Constants.number_of_cuts); i++){
                sumUp = sumUp + x * this.evaluate(x);
                sumDown = sumDown + this.evaluate(x);
                x = x + (1/ Constants.number_of_cuts);
        };
        if      (sumDown == 0)
                return 0;
        else
                return sumUp / sumDown;
        }


/**
 * Halla la ecuación de la recta que pasa por los dos puntos entre los que se encuentra el valor x dado para así
obtener
 * el valor de pertenencia de este.
 */
double evaluate(double x) {
        int i;

        i=0;
        while ((i< this.numPoints ) && (set[i].getX()< x)){
                i++;

        }
        //si el valor dado es menor que el menor valor del fuzzySet entonces su grado de pertenencia es 0.
        if (set[0].getX()> x) { return 0;
                                                        }
        //si el valor dado es mayor que el mayor valor del fuzzySet entonces su grado de pertenencia es 0.
        else if (i== this.numPoints){ return 0;
                                                                }
        //si el valor dado es exactamente uno del fuzzySet.
        else if (set[i].getX()== x){
                        // tengo que buscar el maximo entre los que tengan el mismo x
                        return set[i].getY();
                                                                }
        //ecuación de la recta que pasa por dos puntos.
        else return ((x-set[i-1].getX())*(set[i].getY()-set[i-1].getY()))/ (set[i].getX()-set[i-1].getX()) + set[i-
1].getY();
}


/**
 * Retorna el complemento del  conjunto difuso.
 */
public FuzzySet fuzzyComplement() {
        int i;

        for (i=0;(i< this.numPoints); i++)
                this.set[i].setY(1-(this.set[i].getY()));
        return this;
}
```

```
/**
 * Retorna la intersección  con otherSet.
 */
public FuzzySet fuzzyIntersection(FuzzySet otherSet) {
        int i;
        int j;
        int z;
        FuzzySet newFuzzySet;
        SetPoint[] newSet;

        i = 0;
        j = 0;
        z = 0;

        newSet = new SetPoint[MAX_INITIAL_POINTS];
        if (this.getX(i) < otherSet.getX(j)){
                while ((i< this.numPoints) & (this.getX(i) < otherSet.getX(j))){
              i++;
                }
        }else if (this.getX(i) > otherSet.getX(j)){
                    while ((j< otherSet.numPoints) & (this.getX(i) < otherSet.getX(j))){
                j++;
                    }
                }

        while ((i< this.numPoints) & (j< otherSet.numPoints)){
                if (this.getX(i) < otherSet.getX(j)){
                        newSet[z]= new SetPoint(this.getX(i), Math.min(this.getY(i),
                        otherSet.evaluate(otherSet.getY(j))));
                        i++;
                        }
                else if (this.getX(i) > otherSet.getX(j)){
                        newSet[z]= new SetPoint(otherSet.getX(j), Math.min(otherSet.getY(j),
                        this.evaluate(this.getY(i))));
                        j++;
                        }
                else if ((this.getX(i) == otherSet.getX(j)) & (this.getY(i) >= otherSet.getY(j))){
                        newSet[z] = this.getPoint(i);
                        i++;
                        j++;
                   }
                else if ((this.getX(i) == otherSet.getX(j)) & (this.getY(i) < otherSet.getY(j))){
                        newSet[z] = otherSet.getPoint(j);
                        i++;
                        j++;
                   }
                z++;

        }

        newFuzzySet = new FuzzySet("",newSet,z-1);
        return newFuzzySet;
}


/**
 * Retorna el fuzzySet normalizado.
 */
public FuzzySet fuzzyNormalize() {
        return this.fuzzyScale(this.calculateScaleFactor());
}


/**
 * Retorna el conjunto difuso elevado al valor double dado.
 */
public FuzzySet fuzzyPow(double yValue) {
        int i;
        for (i=0;(i< this.numPoints); i++)
                this.set[i].setY(Math.pow(this.set[i].getY(),yValue));
        return this;
```

```
}


/**
 * Retorna el conjunto difuso multiplicado por el double dado.
 */
public FuzzySet fuzzyScale(double yValue) {
        int i;
        for (i=0;(i< this.numPoints); i++)
                set[i].setY(set[i].getY() * yValue);
        return new FuzzySet ("", this.set, this.numPoints);
}


/**
 * Retorna la unión con otherSet.
 */
public FuzzySet fuzzyUnion(FuzzySet otherSet) {
        int i;
        int j;
        double aux1, aux2, aux3,aux4, aux5, aux6, aux7, x;
        FuzzySet newFuzzySet;

        newFuzzySet = new FuzzySet(this.copy());
        newFuzzySet.appendFuzzySet(otherSet);

        i = 0;
        for (i=0;(i<this.numPoints - 2);i++)
                {
                j = 0;
                while((j < otherSet.numPoints - 2) && this.getX(i+1) >= otherSet.getX(j))
                        {
                        aux2= otherSet.getY(j+1)-otherSet.getY(j);
                        aux3= otherSet.getX(j+1)-otherSet.getX(j);
                        aux4= this.getX(i);
                        aux5= otherSet.getX(j);
                        aux6= this.getY(i);
                        aux7= otherSet.getY(j);
                        if (!(this.getY(i+1)==this.getY(i))) {
                                aux1= (this.getX(i+1)-this.getX(i))/(this.getY(i+1)-this.getY(i));
                                if (!(aux3==(aux1*aux2))) {
                                        x=(aux3*(aux7*aux1-aux6*aux1+aux4)-aux2*aux1*aux5)/(aux3-
                                        aux1*aux2);
                                        newFuzzySet.appendSetPoint(x,0);
                                                                                        }
                                                                                                }
                        else
                                if (!(aux2==0)) {
                                        x = (aux6-aux7)*(aux3/aux2) + aux5;
                                        newFuzzySet.appendSetPoint(x,0);
                                                                }
                        j=j+1;
                        }
                }

                /*newFuzzySet.setY(0,0); */
                for (i=0;(i<newFuzzySet.numPoints );i++)
                {
                        aux1=this.evaluate(newFuzzySet.getX(i));
                        aux2=otherSet.evaluate(newFuzzySet.getX(i));
                        newFuzzySet.setY(i,Math.max(aux1,aux2));
                }
                /*newFuzzySet.setY(newFuzzySet.numPoints-1,0);*/
                return newFuzzySet;
                }


/**
 * Retorna el máximo valor de pertenencia del conjunto difuso.
Retorna el máximo valor de pertenencia del set.
 */
public double getMaxY(){
```

```
        int i;
        double max;

        max=0.0;
        for (i=0;(i< this.numPoints); i++)
                if(set[i].getY() > max) {
                        max = set[i].getY();
                }
        return max;
}


/**
 * Retorna el mínimo valor de pertenencia del fuzzySet.
 */
public double getMinY(){
        int i;
        double min;

        min=1.0;
        for (i=0;(i< this.numPoints); i++);
                if(set[i].getY() < min) {
                        min = set[i].getY();
                }
        return min;
}


/**
 * Retorna el nombre del fuzzySet.
 */
public String getName() {
        return this.name;
}

/**
 * Retorna el setPoint en el indice especificado del fuzzySet.
 */
public SetPoint getPoint(int i) {
        return this.set[i];
}

/**
 * Retorna el valor x de el setPoint en el índice especificado del fuzzySet.
 */
public double getX(int i) {
        return this.set[i].getX();
}

/**
 *Retorna el valor de y en el índice especificado del fuzzySet.
 */
public double getY(int i) {
        return this.set[i].getY();
}

/**
 *Inicializa el fuzzySet.
 */
public void initialize() {
        String sCmd;
        int i;

        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "select_fuzzySet " + "'" + this.idFuzzySet + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        con.first();
        while (!con.isEof()) {
```

```
                this.name= con.getField("name").toString();
                con.next();
                }

        sCmd= "select_FuzzySetPoints " + "'" + this.idFuzzySet + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        i =0;
        con.first();
        while (!con.isEof()) {
                this.set[i] = new SetPoint(con.getField("x").toDouble(),con.getField("y").toDouble());
                con.next();
                i = i + 1;
                }
        this.numPoints = i;
}


/**
 * Retorna si el fuzzySet es vacio; es decir, no tiene puntos.
 */
public boolean isEmpty(){

if (this.numPoints == 0){
        return true;
        }
 else {
          return false;}
          }


/**
 * Retorna la coordenada x del punto de mas a la derecha.
 */
double maxX() {
        return set[this.numPoints - 1].getX();
}


/**
 * Retorna la coordenada x del punto de mas a la izquierda.
 */
double minX() {
        return set[0].getX();
}


/**
 *Busca el setPoint con valores x,y y lo elimina si lo encuentra.
 */
public boolean removeSetPoint(double X, double Y) {
        int i;
        SetPoint point;

 i= 0;
 point = new SetPoint(X,Y);
 while ((i< this.numPoints) & !(this.set[i].equals(point))){
          i++;
 }
 if (this.set[i].equals(point)){
  while( i < this.numPoints ){
           this.set[i] = this.set[i+1];
           this.numPoints = this.numPoints -1;
           i++;
         }
        return true;
 }else
  {
  return false;
  }
 }
```

```
/**
 * Busca el setPoint y lo elimina si lo encuentra.
 */
public boolean removeSetPoint(SetPoint point) {
        int i;

 i= 0;
 while ((i< this.numPoints) & !(this.set[i].equals(point))){
          i++;
 }
 if (this.set[i].equals(point)){
  while( i < this.numPoints ){
           this.set[i] = this.set[i+1];
           this.numPoints = this.numPoints -1;
           i++;
          }
         return true;
 }else
  {
  return false;
  }
 }


/**
 * Retorna el tamaño o el número de puntos del fuzzySet.
 */
public int size() {
        return this.numPoints;
}
```

Clase LFuzzySet

Definición de la clase:

```
public class LFuzzySet extends FuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del lFuzzySet.
 */
public LFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Retorna el valor x del punto más a la izquierda.
 */
double getLeftX() {
        return set[0].getX();
}


/**
 * Retorna el valor y del punto más a la izquierda.
 */
double getLeftY() {
        return set[0].getY();
```

```
}

/**
 * Retorna el valor x del punto más a la derecha.
 */
double getRightX() {
        return set[1].getX();
}


/**
 * Retorna el valor y del punto más a la derecha.
 */
double getRightY() {
        return set[1].getY();
}
```

Clase LeftLinearFuzzySet

Definición de la clase:

```
public class LeftLinearFuzzySet extends LFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del leftLinearFuzzySet.
 */
public LeftLinearFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Método constructor del leftLinearFuzzySet.
 */
 LeftLinearFuzzySet(double leftEdge, double rightEdge) {
        super();
        set = new SetPoint[2];
        set[0] = new SetPoint(leftEdge,0);
        set[1] = new SetPoint(rightEdge,1);
}


/**
 * Esta es la aproximación lineal.
 */

double evaluate(double x) {

        if (numPoints == 2) {
                if (x <= getLeftX())
                        return 0;
                else if (x > getLeftX() & x < getRightX())
                        return (x - getLeftX())/(getRightX() - getLeftX());
                else
                        return 1;
        }
        else
                return 0;
        }
```

Clase SFuzzySet

Definición de la clase:

```
public class SFuzzySet extends LFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método contructor del sFuzzySet.
 */
public SFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Método constructor del sFuzzySet
 */
 SFuzzySet(double leftEdge, double rightEdge) {
        super();
        set = new SetPoint[2];
        set[0] = new SetPoint(leftEdge,0);
        set[1] = new SetPoint(rightEdge,1);
}


/**
 * Esta es la aproximación lineal.
 */
double evaluate(double x) {
        if (numPoints == 2) {
                if (x <= getLeftX())
                        return 0;
                else if (x > getLeftX() & x < getRightX())
                        return (x - getLeftX())/(getRightX() - getLeftX());
                else
                        return 1;
        }
        else
                return 0;
        }

/**
 * Retorna el resultado de evaluar el fuzzySet.
 */
double evaluate(double x) {

        if (numPoints == 2) {
        if (x <= getLeftX())
                return 0;
        else if (x > getLeftX() & x <= getInflectionPoint())
                return 2*((x - getLeftX())/(getRightX() - getLeftX())) * ((x - getLeftX())/(getRightX() -
getLeftX()));
        else if (x > getInflectionPoint() & x <= getRightX())
                return 1 - 2*((x - getRightX())/(getRightX() - getLeftX())) *
                        ((x - getRightX())/(getRightX() - getLeftX()));
        else
                return 1;
        }
        else
        {return 0;
        }
```

```
}


/**
 * Retorna el punto de inflexión del fuzzySet.
 */
double getInflectionPoint() {
        return (getLeftX() + getRightX()) / 2;
}
```

Clase LRFuzzySet

Definición de la clase:

```
public  class LRFuzzySet extends FuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del objeto lRFuzzySet.
 */
public LRFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }
```

Clase PIFuzzySet

Definición de la clase:

```
public class PIFuzzySet extends LRFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor de piFuzzySet.
 */
public PIFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
  * Método constructor del piFuzzySet.
  */
 PIFuzzySet(double leftPoint, double center, double rightPoint) {
        super();
        this.set[0] = new SetPoint(leftPoint,0);
        this.set[1] = new SetPoint(center,1);
        this.set[2] = new SetPoint(rightPoint,0);
        this.numPoints = 3;
}
```

```
/**
 * Retorna el resultado de evaluar el fuzzySet.
 */
double evaluate(double x) {


        if (x <= getLeftX())
                return 0;
        else if (x > getLeftX() & x <= getInflectionLeftPoint())
                return 2*((x - getLeftX())/(getCenterX() - getLeftX())) * ((x - getLeftX())/(getCenterX() -
                getLeftX()));
        else if (x > getInflectionLeftPoint() & x <= getCenterX())
                return 1 - 2*((x - getCenterX())/(getCenterX() - getLeftX())) * ((x - getCenterX())/(getCenterX()
                - getLeftX()));

        else if (x > getCenterX() & x <= getInflectionRightPoint())
                return 1 - 2*((x - getCenterX())/(getRightX() - getCenterX())) * ((x - getCenterX())/(getRightX()
                - getCenterX()));
        else if (x > getInflectionRightPoint() & x <= getRightX())
                return 2*((x - getRightX())/(getRightX() - getCenterX())) * ((x - getRightX())/(getRightX() -
                getCenterX()));
        else
                return 0;
        }



/**
 * Retorna el valor x del punto central.
 */
double getCenterX() {
        return set[1].getX();
}


/**
 * Retorna el punto de inflexión izquierdo.
 */
double getInflectionLeftPoint() {
        return (getLeftX() + getCenterX()) / 2;
}



/**
 * Retorna el punto de inflexión derecho.
 */
double getInflectionRightPoint() {
        return (getCenterX() + getRightX()) / 2;
}


/**
 * Retorna el valor x del punto más a la izquierda.
 */
double getLeftX() {
        return set[0].getX();
}


/**
 * Retorna el valor x del punto mas a la derecha.
 */
double getRightX() {
        return set[2].getX();
}
```

Clase TrapezoidFuzzySet

Definición de la clase:

```
public class TrapezoidFuzzySet extends LRFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del trapezoidFuzzySet.
 */
public TrapezoidFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Método constructor del trapezoidFuzzySet.
 */
public TrapezoidFuzzySet(double zeroLeftX, double oneLeftX, double oneRightX, double zeroRightX) {
        this.set[0] = new SetPoint(zeroLeftX,0);
        this.set[1] = new SetPoint(oneLeftX,1);
        this.set[2] = new SetPoint(oneRightX,1);
        this.set[3] = new SetPoint(zeroRightX,0);
        this.numPoints = 4;
        }


/**
 * Método constructor del trapezoidFuzzySet.
 */
public TrapezoidFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Retorna el resultado de evaluar el fuzzySet.
 */
double evaluate(double x) {

        if (numPoints == 4) {
                if (x <= getZeroLeftX())
                        return 0;
                else if (x > getZeroLeftX() & x <= getOneLeftX())
                        return (x - getZeroLeftX())/(getOneLeftX() - getZeroLeftX());
                else if (x > getOneLeftX() & x <= getOneRightX())
                        return 1;
                else if (x > getOneRightX() & x <= getZeroRightX())
                        return (getZeroRightX() - x)/(getZeroRightX() - getOneRightX());
                else
                        return 0;
        }
        else
        {return 0;
        }
}


/**
 * Retorna el valor x del punto de mas a la izquierda cuyo valor de pertenencia es 1.
 */
double getOneLeftX() {
        return set[1].getX();
}


/**
 * Retorna el valor x del punto de mas a la derecha cuyo valor de pertenencia es 1.
```

```
 */
double getOneRightX() {
        return set[2].getX();
}


/**
 * Retorna el valor x del punto de mas a la izquierda cuyo valor de pertenencia es 0.
 */
double getZeroLeftX() {
        return set[0].getX();
}


/**
 * Retorna el valor x del punto de mas a la derecha cuyo valor de pertenencia es 0.
 */
double getZeroRightX() {
        return set[3].getX();
}
```

Clase RectangleFuzzySet

Definición de la clase:

```
public class RectangleFuzzySet extends TrapezoidFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del rectangleFuzzySet.
 */
public RectangleFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }
```

Clase RectangleFuzzySet

Definición de la clase:

```
public class RectangleFuzzySet extends TrapezoidFuzzySet {
}
```

Métodos públicos:

```
/**
 * Constructor del rectangleFuzzySet.
 */
public RectangleFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }
```

Clase TriangleFuzzySet

Definición de la clase:

```
public class TriangleFuzzySet extends TrapezoidFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del triangleFuzzySet.
 */
public TriangleFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }
```

Clase SingletonFuzzySet

Definición de la clase:

```
public class SingletonFuzzySet extends TriangleFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del singletonFuzzySet.
 */
public SingletonFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }
```

Clase RFuzzySet

Definición de la clase:

```
public  class RFuzzySet extends FuzzySet {
}
```

Métodos públicos:

```
/**
 * Método contructor del rFuzzySet.
 */
 RFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }
```

Clase RightLinearFuzzySet

Definición de la clase:

```
public class RightLinearFuzzySet extends RFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método constructor del rightLinearFuzzySet.
 */
 RightLinearFuzzySet(double leftEdge, double rightEdge) {
        super();
        set = new SetPoint[2];
        set[0] = new SetPoint(leftEdge,0);
        set[1] = new SetPoint(rightEdge,1);
}



/**
 * Método constructor del rightLinearFuzzySet.
 */
 RightLinearFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Retorna el resultado de evaluar el fuzzySet.
 */
double evaluate(double x) {
        if (numPoints == 2){
                if (x <= getLeftX())
                        return 1;
                else if (x > getLeftX() & x < getRightX())
                        return ( getRightX() - x)/( getRightX() - getLeftX());
                else
                        return 0;
        }
        else
        {return 0;
        }
}


/**
 * Retorna el valor x del punto de más a la izquierda.
 */
double getLeftX() {
        return set[0].getX();
}


/**
 * Retorna el valor x del punto de más a la izquierda.
 */
double getRightX() {
        return set[1].getX();
}
```

Clase ZFuzzySet

Definición de la clase:

```
public class ZFuzzySet extends RFuzzySet {
}
```

Métodos públicos:

```
/**
 * Método contructor del zFuzzySet.
 */
 ZFuzzySet(double leftEdge, double rightEdge) {
        super();
        set = new SetPoint[2];
        set[0] = new SetPoint(leftEdge,0);
        set[1] = new SetPoint(rightEdge,1);
}


/**
 * Método contructor del zFuzzySet.
 */

 ZFuzzySet(int idFuzzySet) {
        super();
        this.set = new SetPoint[MAX_INITIAL_POINTS];
        this.idFuzzySet= idFuzzySet;
        initialize();
        }


/**
 * Retorna el resultado de evaluar el fuzzySet.
 */
double evaluate(double x) {

        if (numPoints == 2) {
        if (x <= getLeftX())
                return 1;
        else if (x > getLeftX() & x <= getInflectionPoint())
                return 1 - 2*((x - getLeftX())/(getRightX() - getLeftX())) * ((x - getLeftX())/(getRightX() -
                getLeftX()));
        else if (x > getInflectionPoint() & x <= getRightX())
                return 2*((x - getRightX())/(getRightX() - getLeftX())) * ((x - getRightX())/(getRightX() -
                getLeftX()));
        else
                return 0;
        }
        else
        {return 0;
        }
}


/**
 * Retorna el punto de inflexión.
 */
double getInflectionPoint() {
        return (getLeftX() + getRightX()) / 2;
}


/**
 * Retorna el valor x del punto más a la izquierda.
 */
double getLeftX() {
        return set[0].getX();
}


/**
 * Retorna el valor y del punto más a la izquierda.
 */
double getLeftY() {
        return set[0].getY();
```

```
}

/**
 * Retorna el valor x del punto más a la derecha.
 */
double getRightX() {
        return set[1].getX();
}


/**
 * Retorna el valor y del punto más a la derecha.
 */
double getRightY() {
        return set[1].getY();
}
```

Clase FuzzyModifier

Definición de la clase:

```
public abstract class FuzzyModifier {
        public int numPoints;
        public String name;
}
```

Métodos públicos:

```
/**
 * Método constructor del fuzzyModifier.
 */
public FuzzyModifier() {
        super();
        numPoints = 20;
}


/**
 * Método constructor del fuzzyModifier.
 */
public static FuzzyModifier createFuzzyModifier(int idFuzzyModifier) {

        if (idFuzzyModifier == 1)
                return new NothingModifier();
        else if (idFuzzyModifier == 2)
                return new AboveModifier();
        else if (idFuzzyModifier == 3)
                return new BelowModifier();
        else if (idFuzzyModifier == 4)
                return new ExtremelyModifier();
        else if (idFuzzyModifier == 5)
                return new IntensifyModifier();
        else if (idFuzzyModifier == 6)
                return new MoreorlessModifier();
        else if (idFuzzyModifier == 7)
                return new NotModifier();
        else if (idFuzzyModifier == 8)
                return new PlusModifier();
        else if (idFuzzyModifier == 9)
                return new SlightlyModifier();
        else if (idFuzzyModifier == 10)
                return new SomewhatModifier();
        else
                return new VeryModifier();
}
```

```
/**
 * Expande el fuzzySet recibido.
 */
public FuzzySet expandSet(FuzzySet fuzzySet) {
int i;
double x;
double delta;
FuzzySet newFuzzySet;

        newFuzzySet = new FuzzySet(fuzzySet.copy());
        delta = (fuzzySet.maxX() - fuzzySet.minX()) / (numPoints - 1);
        x = newFuzzySet.minX() + delta;
        for (i=0;(i< this.numPoints - 1); i++){
          newFuzzySet.appendSetPoint(x,fuzzySet.evaluate(x));
          x = x + delta;
        }

        return newFuzzySet;
}



/*
 * Escala el fuzzySet para que al menos un valor tenga valor de pertenecia 1.
 */
public FuzzySet norm(FuzzySet fuzzySet) {
        return fuzzySet.fuzzyNormalize();
}
```

Clase AboveModifier

Definición de la clase:

```
public class AboveModifier extends FuzzyModifier {
}
```

Métodos públicos:

```
/*
 * Aplica el modificador al conjunto difuso dado
   ABOVE  identifica el valor x al cual el máximo valor es alcanzado.
   Todos los valores de pertenencia debajo de este punto son seteados a cero
   y todos los valores de pertenencia arriba de este valor son seteados a 1-y.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;
double max;
        expandSet(fuzzySet);
        max = fuzzySet.getMaxY();
        i = 0;
        while (fuzzySet.set[i].getY() < max){
                        fuzzySet.set[i].setY(0);
                        i++;
        }
        while (i< fuzzySet.size()){
                        fuzzySet.set[i].setY(1-fuzzySet.set[i].getY());
                        i++;
        }
        return fuzzySet;
}
```

Clase BelowModifier

Definición de la clase:

```
public class BelowModifier extends FuzzyModifier {
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado
   BELOW  identifica el valor x al cual el máximo valor es alcanzado.
   Todos los valores de pertenencia arriba de este punto son seteados a cero
   y todos los valores de pertenencia abajo de este valor son seteados a 1-y.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;
double max;

        expandSet(fuzzySet);
        max = fuzzySet.getMaxY();
        i = 0;
        while (fuzzySet.set[i].getY() < max){
                        fuzzySet.set[i].setY(1-fuzzySet.set[i].getY());
                        i++;
        }
        while (i< fuzzySet.size()){
                        fuzzySet.set[i].setY(0);
                        i++;
        }
                return fuzzySet;
}
```

Clase ExtremelyModifier

Definición de la clase:

```
public class ExtremelyModifier extends FuzzyModifier {
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado eleva el factor de pertenencia por un factor de 3.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;
        expandSet(fuzzySet);
        fuzzySet.fuzzyPow(3);
        return fuzzySet;
}
```

Clase IntensifyModifier

Definición de la clase:

```
public class IntensifyModifierextends FuzzyModifier {
}
```

Métodos públicos:

```
/**
  * Aplica el modificador al conjunto difuso dado
  * si(0.0<=y<=0.5) entonces y= 2*y^2
  * si(0.5<y<=1.0)  entonces y= 1-2*(1-y)^2.
  */
public FuzzySet call(FuzzySet fuzzySet) {
int i;

        expandSet(fuzzySet);
        for (i=0;(i< fuzzySet.size()); i++);
                if((fuzzySet.getPoint(i).getY() >= 0.0)&(fuzzySet.getPoint(i).getY() <= 0.5)) {
                        fuzzySet.set[i].setY(2*Math.pow(fuzzySet.getPoint(i).getY(),2));
                 }else
                fuzzySet.set[i].setY(1-2*Math.pow((1-fuzzySet.getPoint(i).getY()),2));
        return fuzzySet;
}
```

Clase MoreorlessModifier

Definición de la clase:

```
public class MoreorlessModifierFuzzyModifier {
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado
 * eleva el factor de pertenencia por un factor de 1/3.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;

        expandSet(fuzzySet);
        fuzzySet.fuzzyPow(1/3);
        return fuzzySet;
}
```

Clase NothingModifier

Definición de la clase:

```
public class NothingModifier{
}
```

Métodos públicos:

```
public FuzzySet call(FuzzySet fuzzySet) {
        return fuzzySet;
        }
```

Clase NotModifier

Definición de la clase:

```
public class NotModifier{
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado
   obtiene el complemento del fuzzySet dado.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;

        expandSet(fuzzySet);
        fuzzySet.fuzzyComplement();
        return fuzzySet;
}
```

Clase PlusModifier

Definición de la clase:

```
public class PlusModifier{
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado
 * eleva el factor de pertenencia por un factor de 1.25
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;

        expandSet(fuzzySet);
        fuzzySet.fuzzyPow(1.25);
        return fuzzySet;
}
```

Clase SlightlyModifier

Definición de la clase:

```
public class SlightlyModifier{
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado
   intensify[plus(A) AND not(very(A)))]
 */
public FuzzySet call(FuzzySet fuzzySet) {
        int i;
```

```
        double aux1;
        double aux2;
        double aux3;

        expandSet(fuzzySet);
        for (i=0;(i< fuzzySet.size()); i++);
          aux1 = Math.pow(fuzzySet.getY(i),1.25);        /** obtiene not(very(A1))*/
          aux2 = 1-Math.pow((fuzzySet.getY(i)),2);       /** obtiene plus(A2)*/
          aux3 = Math.min(aux1,aux2);                   /** obtiene AND(A1,A2)*/
          if((aux3 >= 0.0)&(aux3 <= 0.5)) {             /** obtiene intensify(A)*/
                        fuzzySet.set[i].setY(2*Math.pow(aux3,2));
                }else
                        fuzzySet.set[i].setY(1-2*Math.pow((1-aux3),2));
        return fuzzySet;
}
```

Clase SomewhatModifier

Definición de la clase:

```
public class SomewhatModifier{
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado eleva el factor de pertenencia por un factor de ½.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;

        expandSet(fuzzySet);
        fuzzySet.fuzzyPow(1/2);
        return fuzzySet;
}
```

Clase VeryModifier

Definición de la clase:

```
public class VeryModifier{
}
```

Métodos públicos:

```
/**
 * Aplica el modificador al conjunto difuso dado eleva el factor de pertenencia por un factor de 2.
 */
public FuzzySet call(FuzzySet fuzzySet) {
int i;

        fuzzySet = new FuzzySet(expandSet(fuzzySet));
        fuzzySet.fuzzyPow(2);
        return fuzzySet;
}
```

Clase Repository

Definición de la clase:

```
import java.util.Vector;
 import ar.com.app.types.Constants;
 import ar.com.app.connections.ConnectionSQLAgent;
 import ar.com.app.system.*;
public class Repository {
        public Vector fuzzyVariables;
        public Vector fuzzyValues;
}
```

Métodos públicos:

```
/**
 * Método constructor del repositorio.
 */
public Repository() {
        super();
        initialize();
}

/**
 * Agrega un fuzzyValue al repository.
 */
public void addFuzzyValue(FuzzyValue fuzzyValue) {
        fuzzyValues.addElement(fuzzyValue);
        }

/**
 * Agrega un fuzzyVariable al repository.
 */
public void addFuzzyVariable(FuzzyVariable fuzzyVariable) {
        fuzzyVariables.addElement(fuzzyVariable);
}

/**
 * Retorna el fuzzyValue cuyo identificador coincide con el recibido como parámetro.
 */
public FuzzyValue getFuzzyValue(int idFuzzyValue) {
FuzzyValue fuzzyValue;

        fuzzyValue = new FuzzyValue();
        fuzzyValue.idFuzzyValue = idFuzzyValue;
        return (FuzzyValue) fuzzyValues.elementAt(fuzzyValues.indexOf(fuzzyValue));
        }

/**
 * Retorna el fuzzyVariable cuyo identificador coincide con el recibido como parámetro.
 */
public  FuzzyVariable getFuzzyVariable(int idFuzzyVariable) {
FuzzyVariable fuzzyVariable;

        fuzzyVariable = new FuzzyVariable();
        fuzzyVariable.idFuzzyVariable = idFuzzyVariable;

        return (FuzzyVariable) fuzzyVariables.elementAt(fuzzyVariables.indexOf((fuzzyVariable)));
        }

/**
 * Inicializa el repository.
 */
```

```
void initialize() {
        String sCmd;

        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "selectFuzzyVariables" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        this.fuzzyVariables = new Vector();
        con.first();
        while (!con.isEof()) {
                addFuzzyVariable(new FuzzyVariable(con.getField("idFuzzyVariable").toInt()));
                con.next();
                }

        sCmd= "selectFuzzyValues" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        this.fuzzyValues = new Vector();
        con.first();
        while (!con.isEof()) {
                addFuzzyValue(new FuzzyValue(con.getField("idFuzzyValue").toInt()));
                con.next();
        };

}
```

Clase PageSection

Definición de la clase:

```
import java.awt.Point;
 import ar.com.app.connections.ConnectionSQLAgent;
 import ar.com.app.types.Constants;
public class PageSection {
        public int idSection;
        public String section;
        public Point leftUp;
        public Point rightDown;
        public int seconds;

}
```

Métodos públicos:

```
/**
 * Método constructor de la pageSection.
 */
public PageSection(int idSection) {
        super();
        this.idSection = idSection;
        initialize();
}


/**
 * Inicializa la pageSection.
 */
public void initialize() {
        String sCmd;
```

```
        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "select_Section " + "'" + this.idSection + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }

        con.first();
        while (!con.isEof()) {
                this.section= con.getField("seccion").toString();
                this.leftUp= new Point( con.getField("posX1").toInt(), con.getField("posY1").toInt());
                this.rightDown= new Point( con.getField("posX2").toInt(), con.getField("posY2").toInt());
                this.seconds= con.getField("seconds").toInt();
                con.next();
                }

}
```

Clase Visitor

Definición de la clase:

```
import java.util.Vector;
 import ar.com.app.types.*;
 import ar.com.app.connections.ConnectionSQLAgent;
public class Visitor {
        public int idProfile;
        public String name;
        public String surname;
        public Vector visitorProducts;
        public Vector features;
}
```

Métodos públicos:

```
/**
 * Método constructor del visitor.
 */
public Visitor(int idProfile) {
        super();
        this.idProfile = idProfile;
        initialize();
}


/**
 * Retorna el nivel de aceptación del producto al que hace referencia.
 */
 double acceptanceLevel(Product product) {
        return this.getProduct(product).acceptanceLevel();
}


/**
 * Agrega el feature dado a la lista de features del visitante.
 */
 void addFeature(VisitorFeature feature) {
        this.features.addElement(feature);
}
```

```
/**
 * Agrega el producto dado a la lista de productos del visitante.
 */
 void addProduct(VisitorProduct product) {
        this.visitorProducts.add (product);
}



/**
 * Retorna el feature que esta en la posición i.
 */
 VisitorFeature featureAt(int i) {
        return (VisitorFeature) this.features.elementAt(i);
}



/**
 * Retorna el feature del visitante que corresponde al feature pasado como parámetro.
 */
 VisitorFeature getFeature(VisitorFeature feature) {
        return featureAt(this.features.indexOf(feature));
         }


/**
 * Busca el feature Language en la colleccion de features y lo devuelve (siempre existe).
 */
 int getIdLanguage() {
        int i;

        i=0;
        while (i < this.features.size() & ( this.featureAt(i)).idFeature != 3) { /*el idFeature de lenguaje es 3*/
                i++;
        }
        return (int) this.featureAt(i).value;
}



/**
 * Busca el feature Language en la colleccion de features y lo devuelve (siempre existe).
 */
 double getLanguage() {
        int i;

        i=0;
        while (i < this.features.size() & ( this.featureAt(i)).idFeature != 3) { /*el idFeature de lenguaje es 3*/
                i++;
        }
        return this.featureAt(i).value;
}


/**
 * Retorna el producto de la lista de producto del visitante que corresponde con el producto pasado como
parámetro.
 */
 VisitorProduct getProduct(Product product) {
        return productAt(this.visitorProducts.indexOf(product));
}



/**
 * Incrementa la cantidad de veces que se mostro el producto al vistante.
 */
 void incrementShowedTimes(Product product) {
        this.getProduct(product).incrementShowedTimes();
}
```

```
/**
 * Inicializa el visitante.
 */
public void initialize() {
	String sCmd;
	VisitorFeature feature;
	VisitorProduct product;

	ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
	Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

	sCmd= "select_user " + "'" + this.idProfile + "'";
	if (!con.executeQuery(sCmd)){
		con.close();
	}
	con.first();
	if (!con.isEof()) {
		this.name= con.getField("name").toString();
		this.surname= con.getField("surname").toString();

	}

	sCmd= "profile_Features " + "'" + this.idProfile + "'";
	if (!con.executeQuery(sCmd)){
		con.close();

	}

	con.first();
	this.features = new Vector();
	while (!con.isEof()) {
		feature = new VisitorFeature ( con.getField("featureValue").toDouble(),
		con.getField("idFeature").toInt(), con.getField("featureName").toString());
		this.addFeature(feature);
		con.next();

	}

	this.visitorProducts = new Vector();
	sCmd= "profile_Products " + "'" + this.idProfile + "'";
	if (!con.executeQuery(sCmd)){
		con.close();
	}

	con.first();

	while (!con.isEof()) {
		product = new VisitorProduct (
		con.getField("idProduct").toInt(),con.getField("showedTimes").toInt(),con.getField("visitedTimes")
		.toInt() );
		this.addProduct(product);
		con.next();
		}
}



/**
 * Retorna el producto de la posición i.
 */
public VisitorProduct productAt(int i) {
	return (VisitorProduct) this.visitorProducts.elementAt(i);
}
```

Clase Product

Definición de la clase:

```
import java.util.Vector;
import ar.com.app.types.Constants;
import ar.com.app.connections.*;
public class Product {
        public int idProduct;
        public String name;
        public PageSection pageSection;
        public Vector images;
        public Vector fuzzyVariables; /* los elementos son de la clase variable difusa*/
        public Repository repository;

}
```

Métodos públicos:

```
/**
 * Método constructor del product.
 */
public Product(int idProduct , Repository repository) {
        super();
        this.idProduct = idProduct;
        this.repository = repository;
        initialize();
}


/**
 * Agrega la fuzzyVariable dada a la lista de fuzzyVariables del producto.
 */
public void addFuzzyVariable(FuzzyVariable fuzzyVariable) {
        this.fuzzyVariables.addElement(fuzzyVariable);
}


/**
 * Agrega la imagen dada a la lista de imagenes del producto que se que se utilizarán para mostrar la publicidad.
 */
public void addImage(String image) {
        this.images.addElement (image);
}


/**
 * Retrona true si el objeto recibo es igual al producto, false en caso contrario.
 */
        public boolean equals(Object obj) {
        if ((obj != null) && (obj instanceof Product)) {
            return this.idProduct == ((Product)obj).idProduct();
        }
        return false;
        }


/**
 * Retrona true si el objeto recibo es igual al producto, false en caso contrario.
 */
        public boolean equals(Product product) {
        return (this.idProduct == product.idProduct);
        }


/**
 * Retorna la sección donde debe mostrarse el producto.
 */
public PageSection getSection() {
```

```
        return this.pageSection;
}

/**
 * Retorna el identificador del producto.
 */
public int idProduct() {
        return this.idProduct;
}


/**
 * Inicializa el producto.
 */
public void initialize() {
        String sCmd;

        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "select_product " + "'" + this.idProduct + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        con.first();
        if (!con.isEof()) {
                this.name= con.getField("name").toString();
                }

        sCmd= "product_fuzzyVariables " + "'" + this.idProduct + "'" ;
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        this.fuzzyVariables = new Vector();
        con.first();
        while (!con.isEof()) {
                addFuzzyVariable(this.repository.getFuzzyVariable(con.getField("idFuzzyVariable").toInt()));
                con.next();
                };
        this.images= new Vector();

}


/**
 * Carga todas las imágenes de los banners del producto.
 */
public void loadImages(Visitor visitor) {
        String sCmd;

        ConnectionSQLAgent con = new ConnectionSQLAgent("jdbc:odbc:"+ Constants.ODBC_FUZZY_DSN,
        Constants.ODBC_FUZZY_USER, Constants.ODBC_FUZZY_PASSWORD);

        sCmd= "product_images " + "'" + this.idProduct + "','" + pageSection.idSection  + "','" +
        visitor.getIdLanguage() + "'";
        if (!con.executeQuery(sCmd)){
                con.close();
        }
        con.first();
        while (!con.isEof()) {
                this.images.addElement(con.getField("image").toString());
                con.next();
        }
}


/**
 * Elimina la variable difusa de la lista del producto.
 */
public void removeFuzzyVariable(FuzzyVariable fuzzyVariable) {
        this.fuzzyVariables.removeElement(fuzzyVariable);
}
```

```
/**
 * Elimina una imagen a la lista de imágenes del producto.
 */
public void removeImage(String image) {
        this.images.removeElement(image);
}



/**
 * Setea la sección donde se va a mostrar la publicidad.
 */
public void setSection(PageSection aSection) {
        this.pageSection = aSection;
}


/**
 * Muestra el aviso del producto de acuerdo con la sección dada.
 */
public void showAd(PageSection section, Visitor visitor) {
        this.pageSection = section;
        loadImages (visitor);
        showImages(visitor);
}


/**
 * Muestra las imágenes que hay en images en la sección section
   e incrementa la cantidad de veces que se mostro el producto para este visitante
 */
public void showImages(Visitor visitor) {
        visitor.incrementShowedTimes(this);
        System.err.println(this.name);
}
```

Clase VisitorProduct

Definición de la clase:

```
public class VisitorProduct extends Product {
        public int showedTimes;
        public int visitedTimes;
}
```

Métodos públicos:

```
/**
 * Método constructor del visitorProduct.
 */
public VisitorProduct(int idProduct, int showedTimes, int visitedTimes) {
        super();
        this.idProduct = idProduct;
        this.showedTimes = showedTimes;
        this.visitedTimes = visitedTimes;
}


/**
 * Incrementa la cantidad de veces que la publicidad de este producto fue mostrada al visitante actual.
 */
void incrementShowedTimes() {
        showedTimes=showedTimes + 1;
}
```

```
/**
 * Incrementa la cantidad de veces que la publicidad de este producto fue visitada por el visitante actual.
 */
void incrementVisitedTimes() {
        visitedTimes=visitedTimes + 1;
}
```

Clase VisitorFeature

Definición de la clase:

```
public  class VisitorFeature extends Object {
        public String name;
        double value;
        int idFeature;
}
```

Métodos públicos:

```
/**
 * Método constructor del visitorFeature.
 */
public VisitorFeature(double value,int idFeature, String name) {
        super();
        this.name = name;
        this.value = value;
        this.idFeature = idFeature;
}


/**
 * Retorna true si el feature recibido es igual al que representa el objeto receptor del mensaje.
 */
        public boolean equals(Object feature) {
        return (this.idFeature == ((VisitorFeature)feature).idFeature);
        }


/**
 * Retorna el nombre del feature (ej. Edad).
 */
public String getName() {
        return this.name;
}


/**
 * Retorna el valor asociado al feature que representa el objeto.
 * Ej, si el feature representa la edad podría ser 28.
 */
public double value() {
        return this.value;
}
```

Base de Datos

Tablas:

AdvisorRules (idAdvisor, idRule)

Advisors (idAdvisor, name)

AdvisorSections (idProductAdvisor, idSection)

FuzzyModifier (idModifier, name)

FuzzyRule (idFuzzyRule, idFuzzyValue)

FuzzySet (idFuzzySet, name, idShape)

FuzzySetShape (idShape, name, minPoints)

FuzzyValue (idFuzzyValue, idFuzzyVariable, idModifier, idFuzzySet)

FuzzyVariable (idFuzzyVariable, name, units, lower, upper, idFeature)

FuzzyVariableSets (idFuzzyVariable, idFuzzySet)

PageSections (idSection, section, pos, posY1, posX2, posY2, seconds, isActive, priority)

Parameters (modifierNumPoints)

Product (idProduct, name)

ProductAdvisors (idProductAdvisor, idAdvisor, idProduct, sActive, minDegree, maxAdPerVisitor, adImpressions)

ProductFuzzyVariables (idProduct, idFuzzyVariable)

ProductImages (idProductSection, idProduct, idSection, image, idLanguage)

ProfileFeatures (idProfile, idFeature, featureValue)

ProfileProducts (idProfile, idProduct, showedTimes, visitedTimes, showedDate)

Profiles (idProfile, name, surname)

RulePreconditions (idPrecondition, idFuzzyRule, idFuzzyValue)

SetPoints (idPoint, idFuzzySet, x, y)

VisitorFeature (idFeature, featureName)

VisitorFeaturesDomain (idFeature, valueName, value)

Procedimientos almacenados:

```
CREATE PROCEDURE active_ProductAdvisors
AS
select *
from productAdvisors
where isActive= 1
order by idProductAdvisor
GO


CREATE PROCEDURE active_Sections
AS
select *
from PageSections
where isactive = 1
order by priority
GO


CREATE PROCEDURE advisor_Rules
@idAdvisor int
AS
select *
from AdvisorRules
where idAdvisor = @idAdvisor
order by idRule
GO


CREATE PROCEDURE incrementShowedTimes
@idVisitor numeric,
@idProduct numeric,
@showedDate datetime
AS
select *
from ProfileProduct
where idProfile = idVisitor  and idProduct = @idProduct and showedDate = @showedDate

if @@ROWCOUNT = 0
        update ProfileProduct set  showedTimes = showedTimes + 1
        where idProfile = @idVisitor  and idProduct = @idProduct and showedDate = @showedDate
else
        insert into ProfileProduct (showedTimes, idProfile, idProduct, showedDate)
        values ( 1,  @idVisitor , @idProduct , @showedDate)
GO


CREATE PROCEDURE incrementVisitedTimes
@idVisitor numeric,
@idProduct numeric,
@showedDate datetime
AS
update ProfileProduct set  visitedTimes = visitedTimes + 1
where idProfile = idVisitor  and idProduct = @idProduct and showedDate = @showedDate
GO


CREATE PROCEDURE Product_FuzzyVariables
@idProduct int
AS
select *
from ProductFuzzyVariables
where idProduct = @idProduct
GO
```

```
CREATE PROCEDURE product_images
@idProduct int,
@idSection int,
@idLanguage int
AS
select *
from ProductImages
where idProduct = @idProduct and idSection = @idSection and  idLanguage = @idLanguage
GO



CREATE PROCEDURE profile_features
@idProfile int
AS
select ProfileFeatures.featureValue, VisitorFeature.*
from ProfileFeatures inner join VisitorFeature on ProfileFeatures.idFeature = VisitorFeature.idFeature
where idProfile = @idProfile
GO



CREATE PROCEDURE profile_products
@idProdile int
AS
select *
from ProfileProducts
where idProfile = @idProdile
GO



CREATE PROCEDURE rule_preconditions
@idRule int
AS
select *
from RulePreconditions
where idFuzzyRule = @idRule
order by idPrecondition
GO



CREATE PROCEDURE select_AgeProductRelation
@idProduct int
AS
select *
from AgeProductRelation
where idProduct = @idProduct
GO



CREATE PROCEDURE select_category
@idCategory int
AS
select *
from Category
where idCategory = @idCategory
GO


CREATE PROCEDURE select_FuzzySet
@idFuzzySet int
AS
select *
from FuzzySet
where idFuzzySet = @idFuzzySet
GO
```

```
CREATE PROCEDURE select_FuzzySetPoints
@idFuzzySet int
AS
select *
from SetPoints
where idFuzzySet = @idFuzzySet
order by x
GO


CREATE PROCEDURE select_FuzzyValue
@idLFuzzyValue int
AS
select *, FuzzySet.idShape
from FuzzyValue  inner join FuzzySet on FuzzyValue.idFuzzySet = FuzzySet.idFuzzySet
where idFuzzyValue = @idLFuzzyValue
GO


CREATE PROCEDURE select_fuzzyVariable
@idFuzzyVariable int
AS
select *, VisitorFeature.featureName
from FuzzyVariable inner join VisitorFeature on FuzzyVariable.idFeature =  VisitorFeature.idFeature
where idFuzzyVariable=@idFuzzyVariable
GO


CREATE PROCEDURE select_fuzzyVariableSets
@idFuzzyVariable int
AS
select FuzzySet.*
from FuzzyVariableSets inner join FuzzySet on FuzzyVariableSets.idFuzzySet = FuzzySet.idFuzzySet
where idFuzzyVariable = @idFuzzyVariable
GO


CREATE PROCEDURE select_product
@idProduct int
AS
select *
from product
where idProduct = @idProduct
GO


CREATE PROCEDURE select_ProductAdvisor
@idAdvisor int
AS
select ProductAdvisors.*, Advisors.name
from ProductAdvisors inner join Advisors on  ProductAdvisors.idProductAdvisor = Advisors.idAdvisor
where idProductAdvisor = @idAdvisor
GO


CREATE PROCEDURE select_rule_consequent
@idFuzzyRule int
AS
select idFuzzyValue
from FuzzyRule
where idFuzzyRule = @idFuzzyRule
GO
```

```
CREATE PROCEDURE select_section
@idSection int
AS
select *
from PageSections
where idSection = @idSection
GO



CREATE PROCEDURE select_user
@idProfile int
AS
select *
from profiles
where idProfile = @idProfile
GO



CREATE PROCEDURE selectFuzzyValues
AS
select *
from FuzzyValue
order by idFuzzyValue
GO



CREATE PROCEDURE selectFuzzyVariables
AS
select *
from FuzzyVariable
order by idFuzzyVariable
GO
```

Bibliografía

[1975a] "The concept of a linguistic variable and its application to approximate reasoning, Part I" Inf: Sci 8:301 – 357.

[ARZ01] Marketing en Internet, estrategias básicas para destacarse on-line. Marcelo Arzuaga. Editorial Sudamericana, 2001.

[BAB2001] Fuzzy and Neural Control DISC Course Lecture Notes (October 2001) ROBERT BABU˘SKA.

[Braae and Rutherford, 1978] "Fuzzy relations in a control setting". Kybernetes 7(3): 185 – 189.

[CSGS] Cliente/Servidor Guia de supervivencia (Orfali, Haykey, Edwards)

[FC] Fundamentos de comercialización – Principios y métodos. E. Jerome McCarthy / William D. Perreault, Jr.

[Gil95] A. Gilbert. The role of intelligent agent in the information infrastructure. IBM, January 1995.

[HR95] B. Hayes-Roth. An architecture for adaptative intelligent systems. Artificial Intelligence: Special Issue on Agents and Interactivity

[Fon93] J. Foneman. What's an agent, anyway? A sociological case study. Agents Memo 93-01, 1993.

[FSFL] Klir, G.J. and B. Yuan (1995). Fuzzy sets and fuzzy logic; theory and applications. Prentice Hall.

[Kosko, 1992a] "Neural networks and Fuzzy Systems: A Dinamical Systems Approach to Machine Intellegence". Englewood Cliffs, Nj: Prentice-Hall.

[Kot 1999] El marketing según Kotler Como crear, ganar y dominar los mercados. Philip Kotler. Paidós, 1999.

[Lin and Lee, 1992] "Real time supervised structure/parameters learning for fuzzy neural network". Proc.IEEE Int. Conf. Fuzzy Syst., 1283-1291, San Diego.

[NFS 1996] Neural Fuzzy Systems: A Neuro Fuzzy Synergism to Intelligent Systems (Ching-Teng Lin, C. S. George Lee). Prentice Hall, 1996

[NJ96] M. Wooldrige N.R. Jennings. Software agents. IEEE Review, January 1996.

[Mae95b] P. Maes. Artificial Life metes entertainment: life like autonomous agent. Comunications of the ACM, 38(11), 1995.

[SF96] A. Graesser S. Franklin. Is it an agents, or just a program? A taxonomy for autonomous agents, 1996.