Magíster en Ingeniería del software

Universidad Nacional de La Plata

Facultad de Informática

Tesis

Inteligencia en aplicaciones sensibles a contexto

Director: Dr. Gustavo H. Rossi
Alumno: Ing. José Carlos Diab

Tesis

Abstract

El objetivo de esta tesis es introducir inteligencia en las aplicaciones sensibles a contexto. Este tipo de aplicaciones cambian su conducta dependiendo de la combinación de estímulos que recibe. Esta combinación de estímulos, proveniente de sensores, se denomina contexto. Por ejemplo, en una oficina puedo tener una serie de sensores que midan la intensidad de la luz, cuyos valores pueden ser: oscuro, normal, brillante y el ruido con valores posibles: silencio, moderado y ruidoso. Partiendo de la combinación de estos datos, se podría obtener el contexto "horario_no_laboral", resultado de combinar intensidad de luz = oscuro y ruido = silencio. Una aplicación sensible a contexto podría estar programada para responder al contexto "horario_no_laboral", bloqueando las puertas de la oficina, apagando luces, fotocopiadoras y demás aparatos que no deban usarse en este contexto.

Para que la aplicación reaccione a determinado contexto se lo debe vincular a una acción determinada. Este enlace se establece en una base de conocimientos. Esta consiste en una serie de reglas del tipo Si contextoA y contextoB (ocurren simultáneamente) Entonces hacer Accion1. Por ejemplo, en una aplicación educativa se busca que el sistema brinde acceso a bibliografía específica, dependiendo de la hora y el aula desde donde se accede. En la base de conocimientos habrá una regla que exprese: Si usuario está en "Aula Matemática" y el horario es "Matutino" Entonces brindar acceso a "manuales de matemática nivel secundario". Podría haber otra regla que exprese, Si usuario está en "Aula Matemática" y el horario es "Nocturno" Entonces brindar acceso a "manuales de matemática nivel terciario".

Generalmente, la base de conocimientos es creada y mantenida por un experto (humano) en el dominio de la aplicación. Cuando la dimensión de la base de conocimientos es elevada, resulta inmanejable para un ser humano. El problema se agrava cuando se debe corregir o agregar reglas para que el sistema se comporte de modo distinto. Por ejemplo, cuando se decide abrir el aula de matemáticas en horario "Vespertino" se debe agregar la regla: Si usuario está en "Aula Matemática" y el horario es "Vespertino" Entonces brindar acceso a "manuales de matemática nivel secundario". Las correcciones y agregados de reglas es un proceso delicado, ya que si se introdujera inconsistencias, el sistema se comportaría de una manera indeseada. Por ejemplo: si se introduce la siguiente regla en la base de conocimientos: Si usuario está en "Aula Matemática" y el horario es "Matutino" Entonces brindar acceso a "manuales de matemática nivel terciario". Esta regla estaría en conflicto con la regla que indica que a dicho contexto le corresponde la acción de brindar acceso a "manuales de matemática nivel secundario".

Otros problemas que enfrenta el diseñador de una base de conocimientos son: el tratamiento de los contextos de carácter continuo (por ejemplo el tiempo) y el manejo de la incertidumbre en la información proveniente de sensores.

Los estímulos continuos deben "discretizarse" por algún método para acotar el número de combinaciones posibles.

La solución que se describe en esta tesis ataca el problema de la discretización y la incertidumbre en los estímulos mediante la introducción de lógica difusa en el modelo. La construcción y el mantenimiento de la base de conocimientos se realizan

automáticamente, de esta manera se minimiza la necesidad de intervención humana. Por ejemplo, si la mayoría de las veces que los alumnos entran al aula de matemática en horario matutino buscan “manuales de matemática nivel secundario”, el sistema “aprenderá” esta regla y la volcará en la base de conocimientos. De esta manera el sistema inferirá la necesidad de mostrar una lista de “manuales de matemática nivel secundario” la próxima vez que un alumno ingrese al aula de matemática en horario matutino.

Una característica importante que debe tener el algoritmo de aprendizaje es la “transparencia”. Esto significa que el modelo de reglas construido debe ser comprendido por el usuario.

El mecanismo de software está diseñado con tecnología de objetos, de esta manera permite la evolución independiente de sus partes. Por ejemplo, si en el futuro se decide cambiar el algoritmo de aprendizaje de reglas, las demás partes deberían seguir funcionando sin modificaciones.

Tabla de contenidos

Capítulo I Introducción

1.1 Introducción

En los últimos años, el progreso en la miniaturización de equipos electrónicos dio paso al surgimiento de dispositivos portátiles.

Antes y durante la década del ’70, la computación estaba confinada a círculos empresarios o militares. En la década del ’80, surgieron las computadoras hogareñas que hicieron llegar el poder de la informática a los hogares. A pesar de los recursos limitados de estos dispositivos, por ejemplo: capacidad de memoria principal alrededor de 64 KB, aparatos de almacenamiento secundario con capacidades menores a 100 KB, etc.; el uso de las computadoras por parte de la persona común fue creciendo sostenidamente a lo largo del tiempo.

El advenimiento de la computadora personal mejoró notablemente la relación del hombre con la máquina. Las aplicaciones se hacían cada vez más sofisticadas y útiles para todos los campos. Esto motivó que la computadora dejara de ser un dispositivo auxiliar, para convertirse en un instrumento con un aporte sustancial en el resultado de las empresas y en el nivel de vida de la gente.

Actualmente disponemos de equipos portátiles con recursos similares a las computadoras personales de escritorio.

El software, ha evolucionado notablemente desde los comienzos de la computación. En los primeros años se la consideraba parte del hardware, luego comenzó a tener entidad propia y después de la crisis del software se convirtió en una rama de la ingeniería. Sin embargo, el crecimiento del software como disciplina siempre ha estado por detrás del desarrollo del hardware.

Uno de los desafíos de la ingeniería de software moderna es aprovechar los recursos que aporta el hardware, en los cada vez más pequeños dispositivos móviles, transformándolos en servicios al usuario. Actualmente, aparatos tales como PDA y PalmTop, tienen prácticamente la misma configuración en hardware y software que una computadora de escritorio. Este hecho posibilitó que una persona pueda llevar su computadora personal a cuestas, aunque también se debió solucionar aspectos técnicos auxiliares. El problema que más restricciones imponía en el uso de cualquier dispositivo móvil, era la conexión a periféricos y/o redes mediante cables. El desarrollo de las conexiones inalámbricas fue de gran importancia para el uso generalizado de equipos portátiles, por ejemplo, teléfonos celulares, notebooks, etc.

Luego de tener la posibilidad de llevar la computadora personal en los viajes fuera de la oficina y poder conectarse a redes en forma inalámbrica, surgió otra necesidad en el usuario, esta es recibir la información necesaria en su dispositivo móvil, sin tener que pedirla explícitamente. Por ejemplo, si el usuario es un estudiante e ingresa a la facultad con su PDA, seguramente consultará en ésta horarios de clase, exámenes y otros aspectos relacionados al campo académico; en cambio, cuando esté con su PDA en un club, practicando su deporte favorito, probablemente consulte los días y horarios de los próximos torneos.

La necesidad del usuario de obtener información relevante según la situación particular en la que se encuentre, motivó el desarrollo de las aplicaciones sensibles al contexto.

Dado que el aporte más significativo de los dispositivos móviles es la capacidad de funcionar en diversos lugares, la primera necesidad a satisfacer fue adaptar la

información brindada al usuario, dependiendo del lugar en el que éste se encuentre. De hecho las primeras aplicaciones sensibles al contexto fueron sensibles a la locación.

Nuevamente, surgió otra necesidad por parte del usuario, que es recibir la información adecuada dependiendo no sólo del lugar donde está, sino también del momento, situación anímica, etc.

La creciente necesidad de adaptación obligó a replantear el diseño de las aplicaciones, reconociéndose como un tipo especial, dadas sus características distintivas.

En los primeros momentos de las aplicaciones sensibles al contexto, la adaptación necesaria se hacía modificando el comportamiento de los objetos que componían la aplicación propiamente dicha. Por ejemplo, en un sistema administrativo de una facultad, seguramente existirá una entidad alumno. Para acomodar las distintas conductas que debe tomar el sistema cuando el alumno cambia de locación, se cambia el código, agregándole decisiones en los programas. Este paso tedioso debía realizarse cada vez que se requería que el sistema reaccione a más situaciones, como podría ser el aspecto de contexto temporal.

Uno de los pasos hacia delante más notables, desde el punto de vista del diseño, fue separar la conducta sensible al contexto de la conducta propia de la aplicación. De este modo ambos tipos de aplicaciones pueden evolucionar en forma independiente.

La tendencia actual en aplicaciones sensibles al contexto, es dotarlas de la capacidad de reaccionar a aspectos de contexto no determinísticos, de una manera similar a cualquier otro aspecto de contexto.

1.2 Objetivo de la tesis

El objetivo de la tesis es diseñar una arquitectura de software destinada a inferir aspectos de contexto derivados no determinísticos, en el dominio de aplicaciones sensibles al contexto. El software tiene como característica central la construcción de una base de reglas, cumpliendo de esta manera dos propósitos: predictivo y descriptivo. Este último para brindar transparencia hacia el usuario de las predicciones efectuadas y eventualmente que el mismo pueda ajustar el funcionamiento del software. La base de reglas se construye automáticamente en base a la historia de comportamiento del usuario.

Existen diferentes algoritmos para obtener una base de reglas y una amplia gama de estrategias para inferir la clase correspondiente a un ejemplo dado, aún considerando la misma base de reglas. Por esta razón, se pondrá énfasis en la separación de los conceptos de aprendizaje e inferencia, de este modo el software puede acomodar la evolución de ambos. La única restricción impuesta para los algoritmos de aprendizaje es que generen una base de reglas.

En la tesis se utiliza uno de los algoritmos adecuados en este dominio con el fin de mostrar el uso del software.

Dado el objetivo de la arquitectura de software descripta en esta tesis, la misma se ubica en una capa intermedia entre el tratamiento de sensores y la asignación de servicios al usuario. Por este motivo se reutilizará la arquitectura del framework especializado en aplicaciones sensibles al contexto Ballon. Esta decisión está basada en la separación de las capas de sensado y servicios que caracteriza a dicho framework.

1.3 Aportes de la tesis

Los aportes de la tesis se desprenden de los objetivos planteados y se resume en los siguientes aspectos:

- Proponer una arquitectura de software que permita albergar distintos algoritmos, separando los conceptos de aprendizaje e inferencia de aspectos de contexto derivados no determinísticos. De esta manera los métodos de aprendizaje, poda de reglas e inferencia pueden evolucionar en forma independiente.
- Complementar el otorgamiento de servicios del framework Ballon, hecho en base a aspectos de contexto determinísticos, con la posibilidad de incorporar aspectos de contextos derivados no determinísticos. Las reglas para inferir estos últimos se inducen en forma automática y en base a la historia de comportamiento del usuario.

1.4 Organización de la tesis

La tesis está organizada de la siguiente manera:

Capítulo 2: Aplicaciones sensibles al contexto

Este capítulo es una introducción al tema de la tesis. En el mismo se mencionan las características particulares de una aplicación sensible al contexto. Luego se indican las características deseables que deben tener los algoritmos de clasificación aplicables en este dominio.

Capítulo 3: Aprendizaje y enfoques relacionados

En este capítulo se hace un estudio de los métodos de clasificación más utilizados, se evaluarán sus ventajas y desventajas en relación a su utilización en este tipo de aplicaciones. Al final del capítulo se describirán los desarrollos parecidos en el dominio, luego se plantearán las diferencias y similitudes con el software propuesto.

Capítulo 4: Conceptos básicos

En este capítulo se presentarán los conceptos necesarios para comprender el software desarrollado. Los tópicos que se tocarán son: las diferencias entre lógica difusa y booleana (clásica) y su impacto en los algoritmos de clasificación.

Capítulo 5: Algoritmo

En este capítulo se describirá en detalle uno de los algoritmos de clasificación apropiado para el dominio de aplicaciones sensibles al contexto. Los tópicos que se tocarán en este capítulo son: el proceso de aprendizaje, el aprendizaje incremental, la fuzzificación de los valores ingresados, la optimización y el proceso de inferencia.

Capítulo 6: Arquitectura del sistema

En este capítulo se documentará en detalle los aspectos arquitectónicos de la solución obtenida, haciendo énfasis en el reparto de responsabilidades.

Capítulo 7: Integración con framework Context Aware

En este capítulo se describirá la manera de integrar el software "inteligente" a un framework especializado en aplicaciones sensibles al contexto, de esta manera se complementan las fortalezas de cada uno de los trabajos.

Capítulo 8: Estudio aplicación del sistema

En este capítulo se mostrará con un ejemplo la utilización del software objetivo de la tesis.

Capítulo 9: Conclusiones y trabajos futuros

En este capítulo se hará un resumen de los resultados obtenidos y se los comparará con los objetivos propuestos. Finalmente, se sugerirán líneas de investigación que mejoren y/o complementen el software desarrollado.

Capítulo II Aplicaciones sensibles al contexto

2.1 Introducción

Generalmente las aplicaciones se desarrollan siguiendo un proceso de software, que a grandes rasgos, se inician con la determinación de la frontera del sistema, las entidades externas que aportan o reciben información del sistema y la descripción del mencionado flujo de datos.

Una vez determinado los conceptos anteriores, el análisis y el diseño de la aplicación comienza con los procesos y datos que están dentro del borde especificado. El contexto se considera como un mero "productor" de datos. Estos datos se ingresan al sistema por medio de un evento que genera, en la forma más usual, una persona.

Desde el punto de vista del usuario final, la aplicación es una caja negra, el usuario introduce datos, el sistema los procesa de una forma programada y luego obtiene un resultado.

La pregunta es ¿Qué rasgos caracterizan a las aplicaciones sensibles al contexto que la diferencien de una aplicación normal?

La respuesta que considero más razonable es el modo que las mismas reaccionan a los cambios en el contexto. Las aplicaciones normales reaccionan al contexto de una manera indirecta, es decir alguna persona tiene que tomar la iniciativa de informarle los cambios al sistema. En cambio, las aplicaciones sensibles al contexto deben reaccionar automáticamente a los cambios en el contexto o ambiente donde se utilicen. Más aún, la respuesta debe ser acorde a una situación particular.

Estos rasgos que caracterizan a las aplicaciones sensibles al contexto implica la necesidad de dotarlas de cierto grado de inteligencia, es decir, incorporarle rasgos característicos de los humanos.

Por lo dicho anteriormente se puede inferir una definición intuitiva de lo que llamamos "contexto". El contexto está formado por todos los aspectos que afecten, es decir que sean significativos, en el desempeño de la aplicación desarrollada.

Cuando decimos que este tipo de aplicación debe reaccionar en forma automática a los cambios del ambiente, estamos suponiendo que la aplicación debe "leer" en forma automática el mismo. Este es otro rasgo que diferencia una aplicación común de una aplicación sensible al contexto. Estas últimas incorporan dispositivos de sensado para comunicarse con el ambiente.

En resumen, una aplicación sensible al contexto se caracteriza por tener:

- Mecanismos de sensores: utilizados para conocer y advertir los cambios en el ambiente.
- Mecanismo de decisión: este está compuesto por una serie de reglas del tipo "SI ocurre esta situación ENTONCES tomar esta acción".

Es necesario emular por medio de un dispositivo de software, la inteligencia de un ser humano para identificar cambios en el contexto, determinar la significación del mismo, determinar si la medición es consistente y reaccionar en forma coherente.

Las aplicaciones que reaccionan al cambio de posición del usuario, son las precursoras de este tipo de sistemas. De hecho, los aspectos de contexto que más se

tienen en cuenta en la determinación del "contexto" de una aplicación son: el tiempo y la locación.

El hecho de tener que incorporarle inteligencia a una computadora motiva un desafío y una complejidad extra en el desarrollo de este tipo de aplicaciones.

2.2 Complejidad de las aplicaciones sensibles a contexto

En el punto anterior señalé de una manera resumida los rasgos que caracterizan a una aplicación sensible a contexto y los desafíos que impone. Ahora propongo señalarlos de una manera más explícita.

Consolidación de los aspectos de contexto

El contexto de una aplicación está formado por objetos granulares llamados aspectos de contexto. Estos aspectos de contexto a su vez dependen de sensores. Estos últimos son los dispositivos físicos que están en contacto con el ambiente.

La información proveniente de los aspectos de contextos se agrupa para formar el contexto o situación a la cual el sistema debe reaccionar.

El proceso implica sustituir por medio de una capa de software, las decisiones que tomaría un ser humano ante la ocurrencia de una medición transmitida por el sensor. Por ejemplo, si estamos diseñando una aplicación que reaccione al cambio ambiental de una oficina, podríamos medir la luminosidad exterior y la hora del día. Los valores de dichos aspectos de contexto podrían utilizarse para tomar la decisión de encender o apagar la luz artificial en forma automática.

Por lo tanto, una capa de software debe tener en cuenta distintos aspectos de contexto para determinar el contexto o situación particular.

El mantenimiento

En este tipo de aplicaciones es habitual el agregado, modificación y borrado de los aspectos de contexto que se tienen en cuenta para formar el contexto. Por tal motivo, se debe prestar especial atención a esta característica en el momento de diseño de la aplicación, de modo tal que minimice el costo de mantenimiento. Un método acorde para lograr este objetivo es la utilización del análisis y diseño orientado a objetos.

El esfuerzo hecho en la recopilación de patrones de diseño tiene como objetivo principal minimizar el mantenimiento y promover la reutilización de piezas de software. La premisa más importante en la identificación de un patrón de diseño es "separar lo que cambia". El uso de estos patrones ayuda a mitigar el impacto del mantenimiento.

El tiempo de respuesta

Generalmente, este tipo de aplicaciones necesitan operar en tiempo real. Esta restricción obliga a tener en cuenta el tiempo de respuesta en el momento del diseño de la aplicación. Una respuesta fuera de tiempo, puede tener la misma connotación negativa que una respuesta desacertada del sistema.

Almacenamiento de objetos

En las aplicaciones comunes es habitual alojar los objetos que conforman la aplicación en el servidor. En cambio, en las aplicaciones sensibles a contexto el requerimiento de tiempo de respuesta, limitaciones de procesamiento y memoria, imponen diseñar la aplicación de manera tal que permita la distribución óptima de objetos entre dispositivos móviles del usuario y un servidor.

2.3 Clasificación de aspectos de contextos

El contexto de una aplicación se descompone en entidades atómicas llamadas aspectos de contexto.

Un aspecto de contexto registra una parte del contexto general y está ligado a uno o más sensores. Por ejemplo, se determina que el contexto de una aplicación está integrado por los aspectos de contexto: locación y tiempo. El aspecto locación se conecta a un sensor que identifique un lugar, por ejemplo un dispositivo GPS. El aspecto tiempo se conecta a un sensor que es en realidad un reloj.

Los aspectos de contexto se pueden clasificar según el origen de los datos en:

- Sensados
- Derivados

Los aspectos sensados toman datos directamente desde sensores. Este tipo de aspectos efectúan una elaboración mínima de los datos del sensor. La elaboración que hacen generalmente se refiere a validación de datos de entrada y asignación de una etiqueta. Por ejemplo, el aspecto de contexto “momento del día” toma la hora del sensor (reloj) y lo convierte en un valor semántico “noche”, cuando la hora se encuentra en el rango [20, 24].

Los aspectos derivados hacen una agregación de datos. Estos toman datos procedentes de sensores o de otros aspectos derivados y producen un aspecto resultante mediante la ejecución de un proceso. Este proceso puede ser una simple función matemática, una conjunción de restricciones o incluir un algoritmo complejo relacionado con la inteligencia artificial.

Existen dos tipos de aspectos derivados:

- Determinísticos
- No determinísticos

El valor de un aspecto de contexto determinísticos se consigue mediante un método de cálculo conocido, no hay incertidumbre. Por ejemplo, si el aspecto de contexto “nivel de humo” toma el valor “alto” y el aspecto de contexto “temperatura” toma el valor “alto”, entonces el aspecto de contexto “estado” de la oficina es “anormal”.

El valor de un aspecto de contexto no determinísticos se consigue aplicando un algoritmo de inferencia, existe incertidumbre. Generalmente se utiliza un algoritmo de inteligencia artificial. La mayoría de estos algoritmos están basados en la estadística y operan sobre datos históricos. Por ejemplo, una aplicación es sensible a un aspecto de contexto “situación social”. Este aspecto deriva de aspectos tales como locación, para determinar en que sala está cada persona; el tiempo, para saber si es {mañana, tarde o

noche} y la fecha, para saber si es temporada de trabajo o vacaciones. En base a los aspectos citados el algoritmo puede determinar si la "situación social" toma los valores: {"reunión de trabajo", "reunión por tomar clase" o "reunión irregular"}.

La aplicación podría inferir que el valor del aspecto "situación social" es "reunión irregular" y disparar una alarma al personal de seguridad, ya que podría tratarse de un robo. La inferencia se basa en episodios históricos. Por ejemplo, históricamente se observa que las "reuniones irregulares" se producen cuando el aspecto locación es "centro de cómputos", el aspecto tiempo es "noche" y el aspecto temporada es "vacaciones".

El algoritmo va a utilizar este aprendizaje cuando tenga que inferir el valor del aspecto "Situación social", teniendo como dato el valor de los aspectos locación, tiempo y temporada. Dicho valor se infiere con un cierto grado de incertidumbre.

Esta tesis está enfocada en este tipo de aspectos derivados, es decir los no determinísticos.

2.4 La clasificación y su aplicación en aplicaciones sensibles a contexto

La clasificación es una de las tareas más importantes en el aprendizaje de máquina. La idea de este tipo de algoritmos es asignar una clase a un ejemplo. Las clases deben estar predefinidas. El ejemplo es una tupla que abarca ciertos atributos. En una tupla se tiene uno o más atributos inferidores (variables independientes) y un atributo clase (variable dependiente).

Los algoritmos de clasificación necesitan como datos de entrada, un conjunto de clases definido y un conjunto de ejemplos preclasificados para realizar el aprendizaje supervisado. Por ejemplo, en un negocio de revistas se guarda información del sexo, la edad y el tipo de revistas que compra cada cliente. El tipo de revista sería el atributo clase. Los valores que podría tomar este atributo clase serían, por ejemplo, "revista deportiva", "revista de negocios", "revista de espectáculos". El objetivo de un algoritmo de clasificación sería inferir el tipo de revista (atributo clase) que el cliente va a comprar, sabiendo el sexo y la edad del mismo. La inferencia del algoritmo se basa en las compras hechas por los clientes anteriormente. Supongamos que obtenemos la siguiente regla en base a la compra histórica de revistas:

Si Sexo = "Masculino" y Edad [20, 30] Entonces compra "Revistas deportivas" con una certeza de 70%.

Esta regla aprendida será utilizada por el algoritmo cuando tenga que inferir el tipo de revista que compre el cliente.

¿Cómo se aplica los algoritmos de clasificación en aplicaciones sensibles al contexto? En el punto anterior identifiqué como foco de atención de esta tesis la determinación del valor de un aspecto de contexto derivado no determinístico. Precisamente, se utilizará un algoritmo de clasificación para inferir ese valor.

2.5 Manejo de la incertidumbre y la vaguedad

La incertidumbre está presente en todas las mediciones, todo valor que provenga de sensores estará afectado por un nivel de error.

Cuando se diseña aplicaciones sensibles a contexto se debe prestar atención a la manera de enfrentar los errores de medición, ya que como se indicó anteriormente, el proceso de extracción de datos del ambiente y posterior procesamiento debe hacerse sin la intervención de humana.

La incertidumbre también la tenemos en el valor de la clase inferida. Esto es así porque se utiliza un algoritmo de inferencia. Siguiendo el ejemplo que describe el proceso de clasificación, uno no puede inferir con certeza (forma determinística) qué revista comprará el cliente, aunque la certeza sea 100%.

La incertidumbre que proviene de la frecuencia de ocurrencia de un evento, se tiene en cuenta en los algoritmos de clasificación utilizando los postulados de la probabilidad y la estadística.

Sin embargo, existe otro tipo de incertidumbre que es la vaguedad con la cual está expresado un evento. Es muy importante tener en cuenta esta incertidumbre cuando se trata con atributos cuyo valor es subjetivo. Por ejemplo, volviendo al ejemplo del comprador de revistas, los atributos utilizados para inferir son: sexo y edad (medido en años). El valor de dichos atributos es objetivo, pero qué ocurriría si tomo como atributo predictor el color de cabello del cliente. El color es un atributo inherentemente subjetivo, esto es, si se le pregunta a varias personas por el color de cabello de una persona, las respuestas pueden diferir.

Una posibilidad es omitir este tipo de atributos en los algoritmos de clasificación, sin embargo, estos pueden ser de gran utilidad predictora. Por ejemplo, existen estadísticas realizadas por organismos fidedignos que muestran la influencia del color de un automóvil en la probabilidad de choque.

Otro campo donde la vaguedad está presente es en las emociones humanas. No hay manera de saber con certeza el grado de tristeza o alegría de una persona.

Cuando necesitamos tener en cuenta atributos con definición vaga en los algoritmos de clasificación, debemos utilizar las leyes de la lógica difusa.

2.6 Aspectos deseables del algoritmo de clasificación utilizado

En los puntos anteriores vimos las características y los desafíos involucrados cuando se desarrolla una aplicación sensible al contexto. En base a esta reseña, podemos enumerar las características deseables que debería tener un algoritmo de clasificación adecuado para este tipo de aplicaciones.

Tiempo de respuesta

Las aplicaciones sensibles a contexto se caracterizan por funcionar en tiempo real. Por lo tanto, el tiempo de respuesta es una cuestión crítica. Los algoritmos de clasificación tienen dos etapas bien definidas: el aprendizaje y la inferencia. El tiempo más crítico es la inferencia.

Capacidad de reaprendizaje

Esta capacidad significa que el algoritmo debe adaptar las reglas aprendidas en el momento que llegan nuevas evidencias. Las nuevas evidencias pueden confirmar las reglas aprendidas, pueden refutarlas y también podrían aparecer nuevas reglas. Por ejemplo, recurriendo al ejemplo del cliente que compra revistas, podría ser que un cliente que cumpla las premisas de la regla inferida, esto es: edad entre 20 y 30 años y sexo masculino, decida comprar una revista de espectáculos. La certeza de la regla aprendida disminuye y tal vez pueda ser reemplazada por otra que tenga como valor del atributo clase "Revista de espectáculos", teniendo como antecedente las mismas condiciones que la regla anterior.

El tiempo de aprendizaje suele ser alto en los algoritmos de clasificación. Por lo tanto, representa un desafío mantener actualizadas en línea las reglas que se usarán en la inferencia.

Incorporación de la incertidumbre

En los puntos anteriores vimos las formas de incertidumbre que se presentan en este tipo de aplicaciones. El algoritmo debe funcionar correctamente cuando obtiene evidencia incompleta o con ruido. La evidencia incompleta se da cuando alguno de los atributos utilizados para inferir no tiene valor asociado. Este problema puede ocurrir por error en un sensor o simplemente cuando nuestra fuente de datos no lo informa. En el ejemplo del comprador de revistas, alguien podría no informar su edad.

Cuando el algoritmo recibe evidencia incompleta, debe tomar la parte definida y no desperdiciar información importante.

Como señalamos anteriormente, el algoritmo debe poder funcionar con atributos predictores cuyos valores posibles estén definidos en forma vaga.

Transparencia

Una de las características de las aplicaciones sensibles al contexto es la respuesta automática a los cambios en el ambiente. Sabemos además, que el algoritmo de aprendizaje se va a estabilizar luego de un número potencialmente alto de ejemplos.

Durante el tiempo de estabilización es crítico que el usuario sepa las reglas que aprendió el algoritmo, ya que las mismas serán utilizadas en el proceso de inferencia.

Capítulo III Aprendizaje y enfoques relacionados

3.1 Introducción

En este capítulo se estudiarán los algoritmos existentes para resolver el problema del aprendizaje automático de patrones. En cada uno de ellos se hará un análisis de la conveniencia con respecto a su utilización en las aplicaciones sensibles a contexto.

Al final del capítulo se comentan los trabajos relacionados y describen las diferencias con el enfoque propuesto.

3.2 Clasificación

En la vida diaria clasificamos objetos en forma permanente, usamos para este propósito los sentidos humanos tales como vista, tacto, oído y gusto. Mediante un algoritmo y en base a la percepción recibida por los sentidos, podemos distinguir la diferencia entre un auto y un camión, un pavimento liso de uno rugoso, etc. Este proceso parece sencillo, pero si se pregunta a varias personas cómo hizo, qué algoritmo usó para llegar a obtener el valor inferido, muchas de ellas no lo podrán hacer con detalle y precisión.

El desafío de un algoritmo de clasificación es reproducir el proceso que un humano realiza para distinguir un objeto de otro.

En la práctica existen grandes casos de éxito en la aplicación de técnicas de clasificación basadas en computadoras. Algunos de los ejemplos son [Randon 2004]: el reconocimiento del código postal formado por caracteres escritos a mano, la diagnosis de enfermedades, detección de cáncer de mama en sus etapas tempranas, etc.

La clasificación es un tipo de aprendizaje supervisado. En este tipo de aprendizaje se le suministra un conjunto de ejemplos (clase definida) al algoritmo, y se busca que el mismo construya un modelo que permita encontrar la clase a un ejemplo con clase desconocida.

Tipos de algoritmos de clasificación

La idea central del aprendizaje de máquina es detectar y extrapolar patrones significativos desde los ejemplos recolectados, para construir un modelo del sistema. Existen dos tipos importantes de enfoques en los algoritmos de aprendizaje de máquina [Servente 2002]:

- Algoritmos de caja negra: en este tipo de algoritmos el énfasis está puesto en obtener alto grado de precisión en el proceso de inferencia. La búsqueda de una representación inteligible es un aspecto que no se tiene en cuenta en este tipo de algoritmos. Por lo tanto, los modelos producidos son generalmente complejos y difíciles de entender.

- Algoritmos de caja blanca: en este tipo de algoritmos el énfasis está puesto en una representación inteligible del conocimiento obtenido por medio de los datos.

La precisión en la respuesta es un aspecto secundario. Los árboles de decisión y las redes bayesianas son ejemplos de este tipo de algoritmos.

3.2.1 Algoritmos de clasificación

En esta sección se hará una breve reseña de los algoritmos de clasificación más populares. En cada algoritmo mostrado se hará un pequeño comentario sobre su aplicabilidad en el modelo descripto en esta tesis. La lista no es exhaustiva ya que existen gran cantidad de combinaciones de técnicas.

3.2.1.1 Clasificadores bayesianos

Una red bayesiana representa el conocimiento cualitativo del modelo mediante un grafo dirigido acíclico. Este conocimiento se articula con la definición de relaciones de independencia/dependencia entre las variables que componen el modelo. El aprendizaje cuantitativo se expresa mediante tablas de probabilidad condicionales. Cada nodo de la red debe tener una tabla de probabilidades condicionales vinculadas al nodo padre.

Un clasificador basado en redes bayesianas utiliza la probabilidad a priori del atributo clase y las probabilidades condicionales de los atributos de inferencia, para obtener la probabilidad a posteriori del atributo clase. Estas redes se llaman bayesianas porque se basan en el teorema de Bayes.

El aprendizaje de redes bayesianas comprende dos etapas bien definidas [Kruse 1998],[Acid 1997] :

- Aprendizaje de la estructura
- Aprendizaje de los parámetros

El aprendizaje de la estructura a partir de ejemplos, es la etapa más compleja del aprendizaje. Existen varias metodologías para efectuar esta tarea [Greiner 1998]:

- Estructura dada por elicitación del conocimiento de expertos
- Algoritmos basados en scoring.
- Algoritmos basados en la identificación de independencia condicional

La estructura obtenida por expertos es la más sencilla si existe buena comunicación entre los analistas y los expertos [Nadkami 1999].

Los algoritmos basados en scoring buscan la estructura que maximice o minimice una función.

Los algoritmos basados en independencia condicional obtienen la mejor red, identificando relaciones de independencia condicional entre nodos. Esto lo consiguen por medio de pruebas estadísticas, tales como chi-cuadrado.

El aprendizaje de parámetros se refiere a encontrar las tablas de probabilidad condicional. Simplemente se debe usar las frecuencias relativas obtenidas mediante el conjunto de datos de aprendizaje.

Las estructuras de redes bayesianas más conocidas son [Fernández 2004],[Cheng 1998]:

- Naive Bayes
- TAN
- BAN
- GBN

Naive bayes (*figura 3.1*) es la estructura bayesiana más simple, el nodo de clasificación es el padre de todos los otros nodos. La estructura de la red es fija, por lo tanto el aprendizaje se enfoca en los parámetros cuantitativos. El fundamento de esta simplificación se basa en la hipótesis de que todos los atributos son independientes dado el valor del atributo clase. Esta hipótesis es muy fuerte y poco realista en la mayoría de los casos.

TAN: (tree augmented naive-bayes) Esta estructura (*figura 3.2*) utiliza como base la estructura de Naive-bayes pero permite la conexión entre los nodos utilizados para inferir. La restricción que impone la estructura es que sólo se puede formar árboles entre los nodos.

BAN: (bayesian augmented network) Esta estructura (*figura 3.3*) permite que se formen grafos entre los nodos utilizados para inferir. La única restricción que permanece es que el nodo clasificador debe ser el padre de todos los nodos.

GBN: (generalized bayesian network) Esta estructura (*figura 3.4*) no impone ninguna restricción, el nodo clasificador se trata como un nodo más de la red.

Naive Bayes

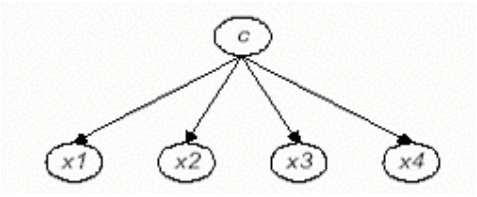


Figura 3.1 Estructura clasificador Naive-Bayes

TAN

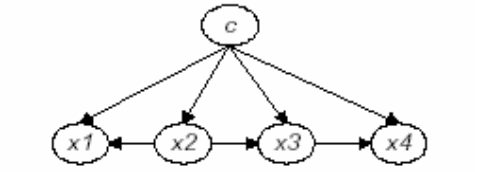


Figura 3.2 Estructura clasificador bayesiano TAN

BAN

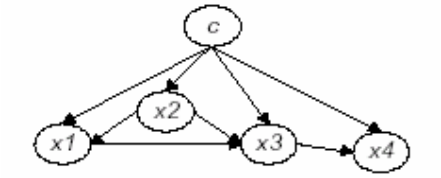


Figura 3.3 Estructura clasificador bayesiano BAN

GBN

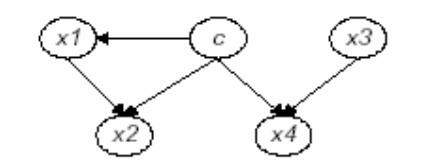


Figura 3.4 Estructura clasificador bayesiano GBN

Comentarios

Los clasificadores TAN, BAN y GBN necesitan un algoritmo que infiera la estructura antes de aprender los parámetros. Esta necesidad implica un tiempo de proceso considerable que no es compatible con las aplicaciones sensibles a contexto.

El clasificador de Naive-Bayes tiene la estructura dada de antemano, por lo tanto no es necesario ningún algoritmo de aprendizaje de estructura. El resultado del clasificador es altamente satisfactorio cuando los nodos no están altamente correlacionados [Rennie 2004]. Numerosos enfoques buscan evitar este problema sin caer en redes más complicadas, por ejemplo: [Storr 2003],[Langseth 2004]. Desde el punto de vista del usuario, resulta poco comprensible un modelo expresado en tablas de probabilidades condicionales. Un problema derivado de tener la estructura fija, es que debe tener en cuenta todos los atributos de inferencia, aún siendo irrelevantes para inferir la clase.

3.2.1.2 Clasificadores basados en distancia

Este tipo de clasificador se basa en el cálculo de la distancia entre los ejemplos de aprendizaje y el ejemplo a clasificar [Orallo 2004c].

La idea central es que la clase de un ejemplo a clasificar, será la misma que la del ejemplo de aprendizaje más cercano.

El concepto de distancia es fundamental para este tipo de algoritmos. Existen varias propuestas para el cálculo de "distancia", la más conocida es la distancia Euclidea. El cálculo de la distancia no sólo existe para valores numéricos, también se puede utilizar cuando los ejemplos están formados por atributos nominales, en este último caso se debe establecer una convención. Por ejemplo, la distancia entre dos valores nominales es 0 cuando son iguales, y 1 en caso contrario.

El algoritmo más popular de este tipo es el K-neighbors. La regla del vecino más próximo simplemente asigna la clase del ejemplo más próximo (1-nearest neighbor). Una variante de este método consiste en asignar la clase mayoritaria entre los K vecinos más próximos. La determinación del valor de K conveniente, cantidad de vecinos a tener en cuenta, es el principal problema de este tipo de algoritmos.

Comentarios

Los algoritmos basados en distancia no construyen un modelo de aprendizaje, la tarea de clasificación se realiza procesando todo el conjunto de ejemplos de aprendizaje. Por este motivo se los llama perezosos, dado que esperan el momento de clasificación para realizar la tarea. La falta de construcción de un modelo, impide considerar a este tipo de algoritmos como una alternativa razonable.

3.2.1.3 Redes neuronales

Este método usa una forma de modelo matemático basado en la operación del cerebro humano. La neurona es el elemento más simple responsable del procesamiento de la información. En el cerebro humano cada neurona se conecta hasta con 200.000 neuronas. La neurona artificial simula las funciones básicas de una natural pero es mucho más simple, tiene menos conexiones [Fiszelew 2001].

La red se estructura en capas. Los distintos algoritmos que corresponden a este método establecen el número de capas que debe tener la red y el cálculo del peso que tiene cada neurona. Como mínimo existen una capa de entrada, una capa de salida y una capa intermedia. Esta última capa de neuronas se denomina oculta (*figura 3.5*). La red ejecuta la clasificación dividiendo los datos en regiones usando hiper planos, los cuales pueden ser no lineales. El proceso de aprendizaje consiste en obtener los pesos asociados a cada neurona.

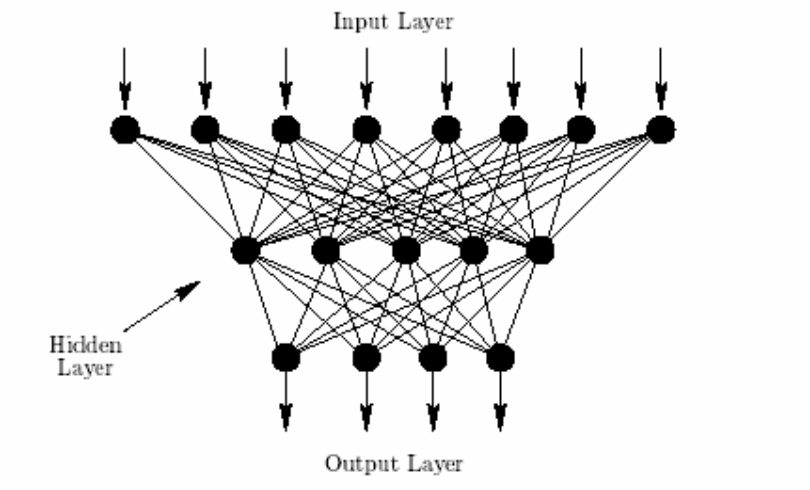


Figura 3.5 Estructura de una red neuronal sencilla

Comentarios

Esta clase algoritmos es del tipo caja negra, ya que el modelo construido es poco comprensible para el ser humano. Por esta razón no es aplicable según las características deseables del algoritmo de clasificación.

3.2.1.4 Árboles de decisión

Este tipo de métodos utiliza el aprendizaje inductivo, esto significa que dado un conjunto de ejemplos preclasificados, establece una generalización para clasificar nuevos ejemplos.

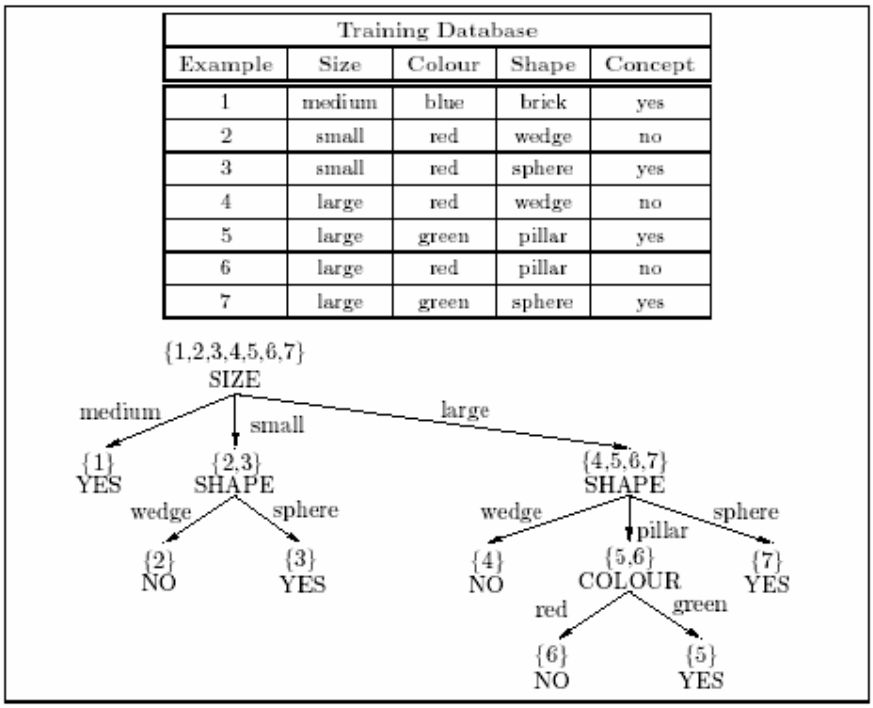
Los algoritmos de esta clase utilizan la estrategia "divide y triunfarás" [Servente 2002] para generar un árbol de decisión a partir de un subconjunto de datos de entrenamiento.

El esquema básico de este tipo de algoritmos consiste en partir el conjunto de ejemplos en conjuntos más puros (ejemplos con igual clase) utilizando una restricción por vez.

El proceso de construcción apunta a encontrar un árbol de decisión que revele una estructura del dominio, y por lo tanto tenga poder predictivo. Para ello necesitamos un número importante de casos en cada hoja, y cada hoja debe tener la menor cantidad posible de casos de distinta clase. La idea es buscar el árbol más pequeño que cubra los ejemplos de aprendizaje.

La *figura 3.6* muestra un árbol de decisión con siete ejemplos de aprendizaje, los cuales se reparten en los distintos nodos del árbol según las restricciones impuestas.

Los algoritmos más representativos de este método de clasificación [Janikow 1996],[Janikow 1994] son: SPRINT, SLIQ, ID3, C4.5.



Training Database

Example	Size	Colour	Shape	Concept
1	medium	blue	brick	yes
2	small	red	wedge	no
3	small	red	sphere	yes
4	large	red	wedge	no
5	large	green	pillar	yes
6	large	red	pillar	no
7	large	green	sphere	yes

Figura 3.6 Ejemplo de clasificador basado en árbol de decisión.

Los árboles demasiado grandes son difíciles de entender (visualmente) porque cada nodo debe ser interpretado dentro del contexto fijado por las ramas anteriores [Marsala 1999], [Marsala 1998], por esta razón el árbol se puede expresar como un conjunto de reglas.

El árbol es una estructura recursiva que permite un sólo padre por nodo, esto implica que existe un solo camino que comunica cada hoja con la raíz. El camino de restricciones presentes desde el nodo raíz a cada hoja se puede transformar directamente en una regla de predicción.

Cuando se clasifica un caso se deben evaluar las restricciones que se presentan desde el nodo raíz hasta la hoja.

Comentarios

Este tipo de algoritmo admite atributos discretos y continuos, tratan correctamente los atributos no significativos, aunque la mayor ventaja es la inteligibilidad del modelo construido. Las reglas de predicción se pueden extraer fácilmente desde el árbol.

Las características expresadas indican que este tipo de algoritmos sea el más adecuado según los aspectos deseables del algoritmo de clasificación expresados en esta tesis.

3.3 Trabajos relacionados

En esta sección se van a describir ciertos trabajos que abordan el tratamiento de aspectos de contexto derivados.

En [Fahy 2004] se propone una arquitectura de software basada en un motor de inferencia y una base de conocimiento. La base de conocimiento se implementa con una base de datos relacional. En la misma se almacenan las reglas necesarias para inferir un aspecto de contexto de alto nivel y se define la conducta sensible al contexto deseada. Cuando existe algún cambio en los aspectos de contexto de bajo nivel, el motor de inferencia aplica las reglas almacenadas en la base de conocimiento. Las reglas deben ser introducidas por el usuario; el framework provee acceso a las mismas mediante el lenguaje SQL.

En [Gu 2004] se describe una arquitectura parecida a la descripta en [Fahy 2004]. Las reglas necesarias para inferir un aspecto de contexto se almacenan en una base de conocimiento de contexto. Un intérprete de contexto toma la base de conocimiento para inferir el aspecto de contexto de alto nivel. A diferencia de [Fahy 2004], la entrega de servicios se realiza en otro módulo y las reglas no se almacenan en una base de datos relacional. Las reglas que conforman la base de conocimiento deben ser ingresadas por el usuario. El framework dispone de una interfaz basada en APIs para actualizar la base de conocimiento de contexto.

En [Korpipaa 2003] los autores presentan un framework que permite el reconocimiento semántico de contextos en tiempo real, teniendo en cuenta el "ruido" en los datos y un entorno de cambio permanente.

Las principales entidades constituyentes del framework son: el administrador de contextos, el servicio de reconocimiento de contextos y la aplicación. El corazón del sistema es el administrador de contextos, el cual controla todo el flujo de la información. Para lograr su cometido, este se apoya en tres dispositivos de software llamados servidor de recursos, servicio de reconocimiento de contextos y el servicio de detección de cambios. El servidor de recursos se encarga de tomar la lectura de sensores, evaluar su consistencia y transformarlo en valores semánticos preestablecidos. Para lograr este cometido utiliza lógica difusa. El servicio de reconocimiento de contextos tiene la misión de inferir, por medio de un algoritmo de clasificación, un aspecto de contexto de alto nivel. El servicio de detección de cambio se utiliza para administrar el mecanismo de avisos. Cuando un contexto de bajo nivel de detalle cambia, le avisa al administrador de contexto para que este a su vez le indique al servicio de reconocimientos que vuelva a inferir el aspecto de contexto de alto nivel.

La aplicación puede operar utilizando aspecto contextos de alto nivel sin necesidad de saber el proceso subyacente.

La estructura del framework permite crear y manejar contextos compuestos y jerarquía de contextos. Entre las tareas más importantes de este framework, se destaca el manejo de la incertidumbre inherente a todo proceso de medición.

El servicio de reconocimiento de contextos está basado en un algoritmo de clasificación de tipo estadístico, como lo es el clasificador naive-bayes. Según los autores, este clasificador funciona bien en la inferencia de contextos por las siguientes razones:

- Robusto: se refiere a su capacidad de obtener resultados razonables aún en presencia de incertidumbre en los valores y/ó ausencia de datos.
- Maneja lógica difusa: permite tomar la incertidumbre en los datos como evidencia virtual.
- Eficiencia computacional: los procesos de aprendizaje e inferencia tienen complejidad lineal respecto del número de casos. Dada su sencillez es muy fácil su implementación.
- No requiere información adicional: esto significa que el algoritmo sólo necesita conocer los datos de la clasificación, no necesita ningún conocimiento previo.

En la *figura 3.7* se muestra la arquitectura general del framework y en la *figura 3.8* se muestra la forma en la cual se obtiene un aspecto de contexto de alto nivel en base a ciertos aspectos de contexto atómicos.

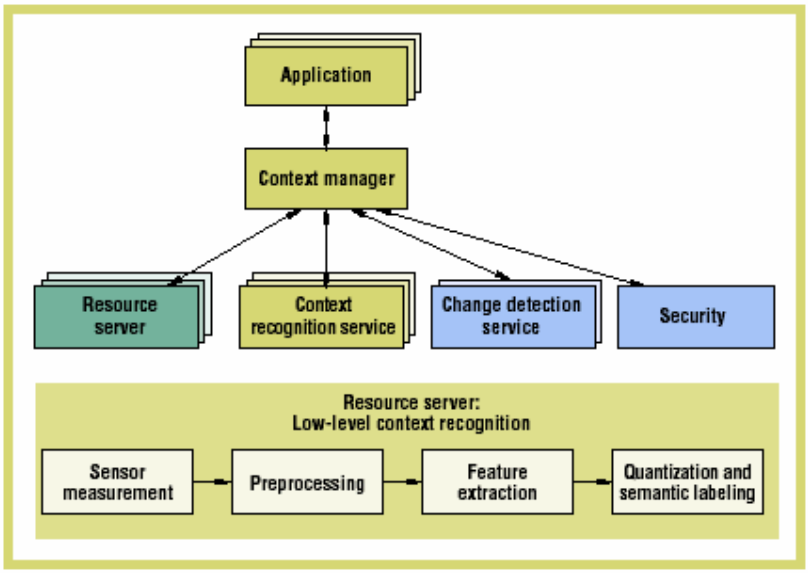


Figura 3.7 Arquitectura general del framework

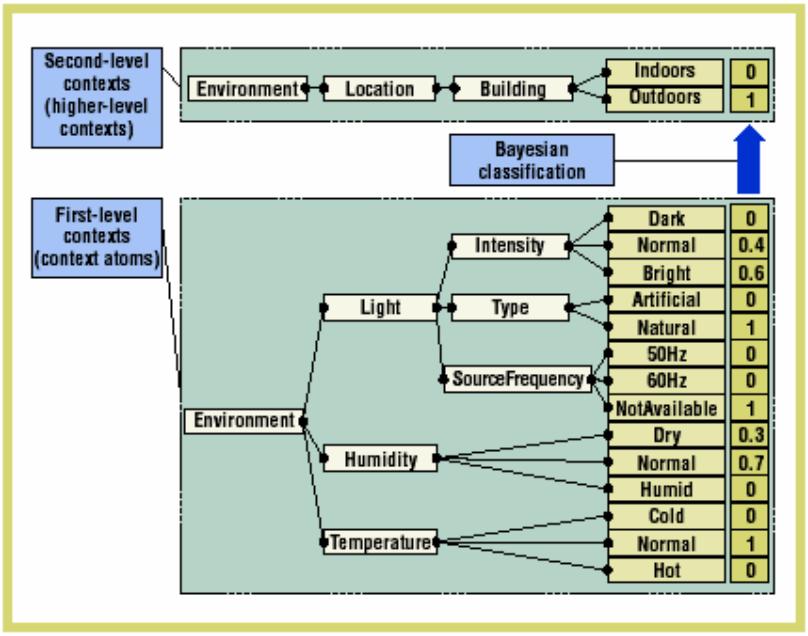


Figura 3.8 Aspectos de contexto atómicos y de alto nivel

En [Cheverst 2003] los autores describen un sistema que tiene una conducta "inteligente" dependiendo del cambio producido en el ambiente. El sistema extrae patrones de comportamiento en base a la historia de uso por parte del usuario. Por ejemplo, se puede determinar que un usuario tiene una reunión en forma regular identificando patrones recurrentes en la historia de contexto del usuario (el contexto está formado por los aspectos de locación, calendario, etc.). La identificación de patrones se realiza utilizando técnicas de aprendizaje de máquina.

El método tradicional para proveer conducta inteligente al sistema, es armar una base de conocimientos formada por reglas predefinidas. Una limitación de este tipo de estrategia es que el usuario debe reconfigurar el sistema cada vez que cambie su rutina. Es evidente que esta tarea puede resultar tediosa y frustrante para el usuario. Por lo tanto, se debe poner el foco en la forma que la máquina aprende reglas de conducta.

La incertidumbre es el problema más importante a resolver cuando se necesita diseñar un sistema que infiera algo. Las fuentes de incertidumbre en este tipo de sistemas son: en el sensado del ambiente y en la interpretación de contextos de alto nivel derivados de contextos de bajo nivel.

Un problema potencial que puede ocurrir en los sistemas predictivos de conducta, es que el sistema puede no comportarse como el usuario desea. Aunque este problema es inevitable, se podría minimizar su impacto incorporando al sistema un dispositivo que explique la conducta inferida.

Este trabajo utiliza un algoritmo basado en árboles de decisión para predecir la conducta del usuario dada su historia. Esto es así porque este tipo de algoritmo es adecuado para mostrar a un humano el conocimiento aprendido por la máquina. La *figura 3.9* ilustra la estructura del sistema.

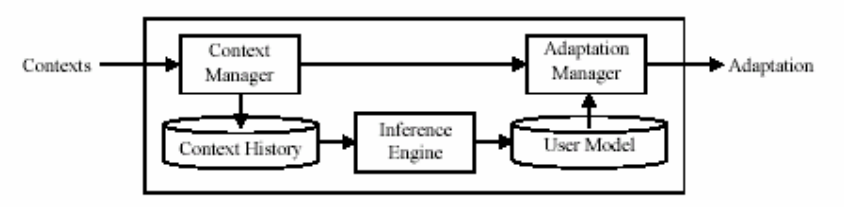


Figura 3.9 Estructura del sistema

3.4 Enfoque propuesto

El enfoque propuesto en esta tesis busca tomar los aspectos positivos de los modelos mencionados.

Los modelos descriptos en [Fahy 2004] y [Gu 2004] tienen como características comunes y destacables la existencia de una base de conocimientos y en forma separada el motor de inferencia. Los problemas que tienen estos modelos son:

- Las reglas para establecer un aspecto de contexto de alto nivel las debe introducir el usuario. Este esquema, además de ser tedioso para el usuario, no considera aspectos de contexto de alto nivel cuyo valor se infiere, es decir, no existe una "fórmula" para calcularlo.
- Los aspectos de contexto se consideran datos. Esto limita la separación de conceptos de adquisición de datos y entrega de servicios.

El modelo expresado en [Korpipaa 2003] tiene las siguientes características importantes:

- La utilización de un algoritmo de aprendizaje e inferencia para inferir el valor de un aspecto de contexto de alto nivel.
- El proceso de reconocimiento de un aspecto de contexto de alto nivel, desde los sensores hasta el aspecto de contexto de alto nivel.
- La utilización de lógica difusa. Esta característica permite utilizar aspectos de contexto con dominio continuo o discreto.

Los puntos débiles de este modelo son:

- La utilización del algoritmo Naive Bayes para inferir un contexto de alto nivel. Este modelo no genera reglas de inferencia, por lo tanto es poco inteligible para los usuarios. La precisión del método no es un punto fuerte, la eficacia depende del cumplimiento de las fuertes simplificaciones propuestas. La simplificación más fuerte es la hipótesis de independencia condicional entre las variables utilizadas para inferir. El algoritmo necesita como parámetro las probabilidades a priori de las clases. Generalmente se asume una distribución uniforme, lo cual puede no ser la realidad.
- La estructura no permite utilizar otro algoritmo de aprendizaje e inferencia.

Los puntos fuertes expresados en [Cheverst 2003] son los siguientes:

- La utilización del algoritmo árbol de decisión para el aprendizaje. Este tipo de algoritmo es adecuado para obtener reglas de comportamiento inteligibles que luego se utilizarán en la inferencia.

- El reconocimiento de una base de datos que almacena la historia y otra que almacena las reglas de comportamiento.
- La incorporación de incertidumbre en el modelo mediante la utilización de lógica difusa.

Los puntos débiles de este modelo son:

- La arquitectura propuesta está limitada a un algoritmo de aprendizaje e inferencia.
- El modelo adolece del mismo defecto que [Fahy 2004], dado que no se reconoce una capa de servicios.
- Los aspectos de contexto se tratan como simples datos.

El enfoque propuesto tiene las siguientes características:

- Utiliza una base de conocimiento modelada como objeto: esto permite agregarle comportamiento inherente a la misma, por ejemplo, distintos métodos de poda e inferencia.
- Los aspectos de contexto se modelan como objetos: esto permite incorporarles conducta.
- Utiliza un repositorio de casos modelado como objetos: los casos de aprendizaje se almacenan en una estructura con funcionalidad propia. Este diseño permite separar los ejemplos de su utilización. El objeto brinda servicios de conteo a los distintos algoritmos de aprendizaje.
- Utilización de lógica difusa: esta característica permite adoptar aspectos de contexto con dominio continuo o discretos y modelar la incertidumbre en sus dos formas: vaguedad y ocurrencia incierta de un evento.
- Utilización de una capa de servicios: el enfoque propuesto propone brindar los servicios correspondientes, reutilizando la capa de servicios del framework Balloon.
- Sugiere la utilización de un algoritmo de aprendizaje basado en árbol de decisión incremental: esta característica permite cumplir con los requerimientos de inteligibilidad y mantenimiento en línea de las reglas de comportamiento que gobiernan la inferencia de los aspectos de contexto de alto nivel.
- El modelo se comporta como un sensor de alto nivel: esta característica permite considerar a los aspectos de contexto de alto nivel de la misma manera que uno de bajo nivel en la capa de servicios.

Capítulo IV Conceptos básicos

4.1 Introducción

En este capítulo se mencionan los conceptos necesarios para entender el desarrollo de los capítulos posteriores.

Uno de las características importantes del algoritmo descrito en esta tesis es el empleo de lógica difusa, por lo tanto se destinará gran parte del capítulo a este tema.

De acuerdo a los conceptos expresados en el capítulo anterior, uno de los algoritmos que más se ajusta a los requerimientos impuestos es el árbol de decisión, debido a esta razón se estudia las variantes de este tipo de algoritmo.

El software construido en esta tesis almacena el conocimiento obtenido en una base de reglas, por esta razón se incluyen conceptos de reglas de asociación. En la sección destinada a este tipo de algoritmo se describirán generalidades, las diferencias utilizando lógica booleana y difusa, y luego se mencionará su utilización en problemas de clasificación.

4.2 Lógica difusa

La lógica difusa surge como una extensión de la lógica clásica o booleana (los términos lógica clásica, lógica tradicional y lógica Booleana se tomarán como sinónimos). En la lógica tradicional se expresa que una proposición puede ser verdadera o falsa. Utilizando notación numérica, el resultado de la evaluación de una proposición sería {0,1}; 0 significa falsedad absoluta y 1 veracidad absoluta. El conjunto de valores posibles lo integran sólo dos valores.

La lógica tradicional es aplicable cuando la precisión es posible y necesaria. Es posible cuando existe un método objetivo para la evaluación de una proposición. Por ejemplo, la proposición “El precio del producto A es $20,50”, controlando el ticket del supermercado puedo determinar con absoluta certeza la veracidad o no de la expresión. En cambio, si quisiera evaluar la proposición “El producto A es caro” surgiría la subjetividad. Esto implica que la proposición sólo puede evaluarse con un cierto grado de certeza. En este caso decimos que el grado de veracidad es un número real que varía en el rango [0,1].

Un caso muy común en el cual se emplea lógica difusa, es cuando se requiere evaluar el clima. Por ejemplo, si preguntamos a un grupo de personas sobre el estado del clima, obtendríamos respuestas diferentes según la subjetividad de cada persona.

Existen casos en los cuales la precisión no es necesaria ni deseable, por ejemplo, si necesito un sistema que tome una decisión dependiendo del momento del día, siendo sus valores posibles {mañana, tarde, noche}, es inútil saber la hora con exactitud. Además generalmente se requiere que la respuesta sea gradual, por lo tanto la definición de rangos precisos no daría los resultados esperados.

La medición es un proceso que intrínsecamente está sujeto a error, este error aumenta cuando la magnitud se mide en forma indirecta. Por ejemplo, el velocímetro de un auto moderno, es un voltímetro (sensor) calibrado en KM/H. En las aplicaciones sensibles al contexto se utilizan sensores para conocer el valor de cada uno de los aspectos de contexto que forman el contexto del sistema. La incertidumbre del valor medido por el sensor debería tenerse en cuenta en los algoritmos de procesamiento de señales.

El interés en la lógica difusa radica en su utilización como técnica para incorporar la ambigüedad e imprecisión del conocimiento humano dentro de los algoritmos de aprendizaje e inferencia.

4.2.1 Fuzzy sets

La noción de conjunto difuso fue introducida por Lofti Zadeh en 1965, quien luego desarrolló el cálculo aproximado a partir del concepto mencionado anteriormente [Randon 2004].

Un set (conjunto) es una colección finita o infinita de objetos, los cuales son miembros o elementos del conjunto.

En la teoría clásica de conjuntos, un objeto pertenece o no a un conjunto; en cambio en la teoría de fuzzy sets, un objeto puede pertenecer parcialmente a un conjunto. El grado de pertenencia de un valor con un fuzzy set queda establecido mediante una función de membresía. El máximo valor de membresía es 1 y el mínimo es 0. Por lo tanto, el concepto de fuzzy set es una extensión de un conjunto clásico, en el cual un conjunto clásico es un caso especial de fuzzy set.

En la *figura 4.1* se muestra una variable base “height” con dominio continuo y la variable lingüística “HEIGHT” definida sobre la misma. La variable lingüística toma los valores lingüísticos {“Very short”, “Short”, “Normal”, “Tall”, “Very tall”}. El significado de cada valor lingüístico se obtiene mediante una regla semántica que involucra restricciones difusas. Por lo tanto, una variable lingüística toma valores lingüísticos, los cuales representan “etiquetas” de los conjuntos difusos (fuzzy sets) definidos.

En la *figura 4.2* se muestra el cálculo del grado de membresía para la variable lingüística Height y el valor lingüístico “Tall Men”, dado el valor de la variable base.

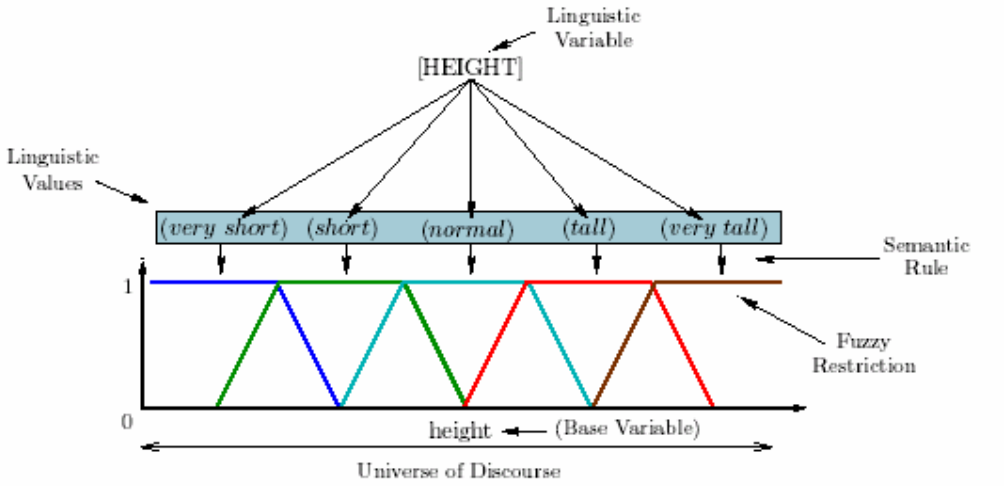


Figura 4.1 Conversión variable base-variable lingüística

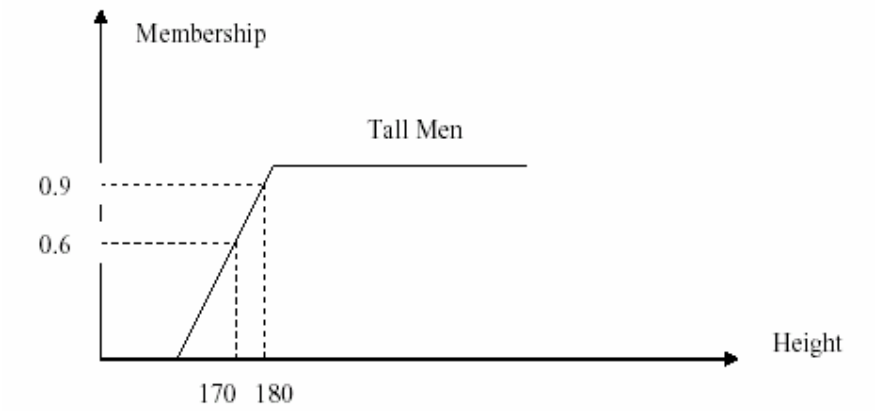


Figura 4.2 Cálculo del grado de membresía para valores de la variable base.

4.2.2 Funciones de membresía

Una función de membresía podría ser cualquier función que, dado el valor de la variable medida, devuelva un real en el rango [0,1]. Dicho número real representa el grado de membresía de un objeto con un conjunto difuso. La función más adecuada depende de las necesidades de cada modelo en particular [Randon 2004].

Las formas de función más utilizadas son triángulo (*figura 4.3*) y trapezoidal (*figura 4.4*).

Triangular Membership Functions

$$\mu(u) = \begin{cases} 0 & if\ u < a \\ \dfrac{u-a}{c-a} & if\ u \in [a,c] \\ \dfrac{b-u}{b-c} & if\ u \in [c,b] \\ 0 & if\ u > b \end{cases}$$

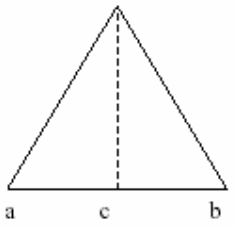


Figura 4.3 Función de membresía triángulo.

Trapezoidal Membership Functions

$$\mu(u) = \begin{cases} 0 & if\ u < a \\ \dfrac{u-a}{m-a} & if\ u \in [a,m] \\ 1 & if\ u \in [m,n] \\ \dfrac{b-u}{b-n} & if\ u \in [n,b] \\ 0 & if\ u > b \end{cases}$$

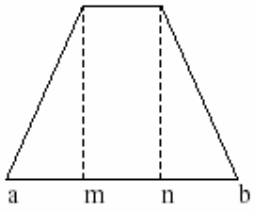


Figura 4.4 Función de membresía trapezoidal.

Una restricción que se va a imponer a las funciones de membresía, es que dado el valor del dominio de la variable base, la sumatoria de los grados de membresía con los fuzzy sets definidos debe ser igual a 1. Por ejemplo, la temperatura de 25 ° C es Cálido con 0,7 grado de membresía y es Templado con 0,3 grado de membresía. Esta restricción, que es muy popular cuando se combina lógica difusa y probabilidad, obedece a una cuestión de lógica. Como se indicó anteriormente, la lógica difusa modela la vaguedad con la cual se describe una cosa. Este modelo debe ser completo.

4.2.3 Operaciones fuzzy

Los fuzzy sets, al igual que los sets clásicos, tienen operaciones básicas asociadas.

Considerando: A y B fuzzy sets definidos en el universo U; $\mu_A(u)$, $\mu_B(u) \in [0,1]$ sus correspondientes funciones de membresía, se definen las siguientes operaciones básicas [Nath 2004], [Mendel 1995].

Intersección $\mathbf{A} \cap \mathbf{B}$: $\mu_{A \cap B}(u)$ = T(A(u),B(u)). $\forall$ u $\in$ U. T(.) se denomina triangular norm. A(u) significa grado de membresía del valor u con el fuzzy set A, B(u) significa grado de membresía del valor u con el fuzzy set B.

Union $\mathbf{A} \cup \mathbf{B}$: $\mu_{A \cup B}(u)$ = TC(A(u),B(u)). $\forall$ u $\in$ U. TC(.) se denomina triangular conorm. A(u) significa grado de membresía del valor u con el fuzzy set A, B(u) significa grado de membresía del valor u con el fuzzy set B.

Complemento de A: 1 – A(u). $\forall$ u $\in$ U. A(u) significa grado de membresía del valor u con el fuzzy set A.

Las propiedades que deben cumplir los operadores T-norms son las siguientes:

Sea $T:[0,1]\times[0,1]\rightarrow[0,1]$. T es un operador T-norm si y sólo si para todo x,y,z ∈ [0,1] cumple las siguientes propiedades:

- Propiedad conmutativa T(x,y) = T(y,x)
- Propiedad de monotonía T(x,y) <= T(x,z) si y <= z
- Propiedad asociativa T(x,T(y,z)) = T(T(x,y),z)
- Elemento identidad T(x,1) = x
- Elemento nulo T(x,0) = 0

Sea $T:[0,1]\times[0,1]\rightarrow[0,1]$. T es un operador T-conorm si y sólo si para todo x,y,z ∈ [0,1] cumple las siguientes propiedades:

- Propiedad conmutativa T(x,y) = T(y,x)
- Propiedad de monotonía T(x,y) <= T(x,z) si y <= z
- Propiedad asociativa T(x,T(y,z)) = T(T(x,y),z)
- Elemento identidad T(x,1) = 1
- Elemento nulo T(x,0) = x

Los T-norms más populares son:

T(x,y) = min(x,y)

T(x,y) = x*y Este operador también se lo llama T-norm probabilístico

Los T-conorms más populares son:

T(x,y) = max(x,y)

T(x,y) = x+y – x*y Este operador también se lo llama T-conorm probabilístico

Existen otros operadores que tienen una aplicación específica. La enumeración exhaustiva de ellos está fuera del alcance de esta tesis. El lector interesado puede tener un estudio exhaustivo en [Mendel 1995].

Las diferencias más notables con los sets clásicos son:

- Los fuzzy sets no cumplen con la ley de los medios excluidos. Por ejemplo: $A \cup \overline{A} <> U$. Siendo U el universo del discurso.
- Los fuzzy sets no cumplen con la ley de la contradicción. Por ejemplo: $A \cap \overline{A} <> \{\}$ (conjunto vacío).

Estas diferencias se deben al solapamiento que se puede presentar entre los fuzzy sets definidos. Por ejemplo, el valor de temperatura 25 °C pertenece a dos fuzzy sets distintos, aunque con distinto grado de membresía.

4.3 Vaguedad vs. probabilidad

Los conceptos de lógica difusa y de probabilidad son tomados como intercambiables por algunos autores, sin embargo tienen diferencias notorias.

La lógica difusa modela la vaguedad en la descripción de una cosa [Pan 1998]. La contraposición a la vaguedad es la certeza en la descripción de una cosa. Esto implica que la lógica difusa modela la subjetividad, mientras que la lógica clásica modela la objetividad. Por ejemplo, una temperatura de 25 ° C es una medición objetiva, es decir, existe un método objetivo de reproducir la medición. En cambio, decir que la temperatura es alta es una medición subjetiva, cada persona tiene su propia interpretación de la temperatura.

La probabilidad mide la frecuencia de ocurrencia de un evento. En este caso la clave es la ocurrencia, es decir, la cantidad de veces que ocurre un evento. Utilizando el ejemplo de la temperatura, la frecuencia relativa (estimador de la probabilidad) del evento Temperatura = 25 °C se calcula como la cantidad de veces que la temperatura es 25 °C, sobre la cantidad de mediciones realizadas.

Por lo dicho anteriormente, decir que la temperatura es "Cálida" con 0,7 grado de membresía, no implica que el evento temperatura "Cálida" tenga 0,7 de probabilidad. La recíproca tampoco es correcta. Esto indica que la probabilidad y la lógica difusa son dos conceptos distintos, cuya característica común es que tanto el grado de membresía como el valor de probabilidad, es un número real que varía en el rango [0,1].

4.4 Árbol de decisión

Un clasificador basado en árbol de decisión es una estructura jerárquica en la cual se aplica una prueba a uno o varios valores de atributos en cada nivel. El resultado de cada prueba puede ser una hoja, la cual contiene sólo una clase, o un nodo interno. El nodo interno es un nodo de decisión que especifica una prueba sobre los valores de un atributo, formando de esta manera una rama del árbol.

La clasificación se efectúa moviéndose desde la raíz hasta las hojas, se formulan una serie de preguntas en orden jerárquico y la decisión final sobre la clase se toma teniendo en cuenta las respuestas a las preguntas anteriores. En forma similar, la relación de una hoja con el árbol al que pertenece se puede describir como una jerarquía de decisiones (ramas), que desde el nodo raíz llega a la rama de la cual cuelga la hoja.

La estructura jerárquica de los árboles de decisión es una de sus características más salientes, la mayoría de los árboles se dibujan de arriba hacia abajo (desde la raíz a las hojas).

Otra característica distintiva de los árboles de clasificación es su flexibilidad. Esta característica proviene de la habilidad de examinar los efectos de las variables predictivas, de a una por vez.

Un clasificador basado en árbol de decisión es un clasificador que se construye jerárquicamente, este compara los datos con una selección apropiada de atributos. La selección de atributos se hace evaluando las distribuciones estadísticas o la separación entre las clases.

La construcción de árboles de decisión se denomina también árbol de inducción porque parte de un conjunto de ejemplos para obtener una teoría sobre el dominio. Cada ejemplo es un objeto que está completamente descrito por un conjunto de atributos y la clase a la que pertenece. Un árbol de decisión contiene cero o más nodos internos y uno

o más nodos hojas. Los nodos internos tienen dos o más nodos hijos. Cada nodo hoja tiene una clase dominante.

Los árboles de clasificación se usan para predecir el grado de pertenencia de los objetos (ejemplos) con los valores de la variable categórica dependiente (la clase).

4.4.1 Construcción de árboles de decisión

La mayoría de los algoritmos de construcción de árboles de decisión procede en una manera arriba – abajo, es decir desde la raíz a las hojas.

El algoritmo arriba – abajo procede de la siguiente forma:

- Si se cumplen los criterios de parada terminar el proceso, el nodo creado se convierte en hoja.
- Si no se cumplen las condiciones de parada, entonces evaluar las posibles particiones (atributos) mediante una métrica y un criterio de selección.
- Elegir la mejor partición (según el valor de la métrica elegida) y crear tantos nodos hijos como valores tenga el atributo elegido.
- Continuar el proceso en forma recursiva con cada nodo hijo creado.

El árbol de decisión impone una división en el espacio formado por los atributos utilizados para clasificar, por lo tanto, las particiones hechas se pueden representar geométricamente como una colección de hiperplanos o regiones. Por esta razón, las divisiones se interpretan como hiperplanos (más de dos dimensiones), los atributos como dimensiones y los objetos como puntos.

4.4.2 Criterios de selección

El objetivo principal del criterio de selección es separar los ejemplos en nodos hijos más puros. Por este motivo, la mayoría de los criterios propuestos se basan en métricas derivadas de las frecuencias relativas de las clases. Por ejemplo, si en un nodo tenemos un 50% de ejemplos clase a y un 50% clase b, aplicamos un criterio que divida el nodo en dos nodos hijos cuya pureza sea 100% en cada uno, dicho criterio sería excelente [Orallo 2004b].

Basándose en la idea de buscar particiones que consigan nodos más puros, se propusieron numerosas métricas, las más utilizadas son: Gini y Gain. Estos criterios buscan la partición s con la menor I(s).

$$\text{I(s)} = \sum_{j=1..n} p_j * f\left(p_j^1, p_j^2, ..., p_j^c\right)$$

En el cual n es la cantidad de nodos hijos, p_j es la frecuencia relativa de "caer" en el nodo j, p_j^1 es la proporción de ejemplos de la clase 1 en el nodo j, p_j^2 es la proporción de ejemplos de la clase 2 en el nodo j, y así para las c clases. Bajo esta fórmula general, cada criterio implementa una función $f\left(p_j^1, p_j^2, ..., p_j^c\right)$ distinta.

Gini: $1 - \sum_{i=1..c} (p_i)^2$

Entropía: $\sum_{i=1..c} p_i * \log(p_i)$

Las funciones se denominan funciones de impureza. Existen variantes de estos criterios, por ejemplo Gain Ratio, y muchos criterios que no se ajustan a la fórmula general. Entre estos últimos se ubica el MDL (niminum description length).

4.4.3 Criterio de parada

El criterio de parada en la construcción de un árbol de decisión se refiere a las condiciones utilizadas para determinar el "tamaño correcto" del mismo. El criterio de parada busca un equilibrio entre la cantidad de nodos del árbol y la precisión en la etapa de clasificación.

Los criterios utilizados para detener el algoritmo son los siguientes:

- Se alcanzó un determinando umbral de frecuencia relativa (ingresado como parámetro) en alguna clase de la partición.
- No existen más ejemplos para dividir el nodo o el nodo tiene una cantidad de ejemplos despreciable (cantidad ingresada como parámetro).

4.4.4 Poda en árboles de decisión

El objetivo de los algoritmos de poda en los árboles de decisión es simplificarlos, de esta manera se reduce la complejidad en el proceso de clasificación. Otro de los objetivos de la poda es disminuir el efecto de sobreajuste. Este problema es una desventaja inherente a los árboles de decisión. Este efecto significa que el sistema de clasificación pierde precisión en la inferencia al adherirse totalmente a una muestra, la cual puede no ser significativa o representativa del dominio. Dado que el sobreajuste es un problema común a todos los métodos basados en árboles de decisión, este punto concentra la atención de los investigadores. En la bibliografía especializada se propusieron una gran cantidad de algoritmos que varían sustancialmente en el enfoque utilizado. Algunos métodos proceden desde la raíz hasta las hojas (Top-Down) y otros métodos proceden en forma inversa (Bottom-up), estos últimos son los más utilizados. Existen algoritmos que reutilizan el conjunto de aprendizaje (ya usado en la etapa de construcción del árbol) y otros que utilizan un conjunto adicional de datos (distinto al conjunto de aprendizaje), llamado conjunto de prueba. Este último conjunto se supone que brinda menos sesgo, aportando una precisión predictiva superior al árbol de decisión.

Algunos de los métodos de poda son: Poda por reducción de error, poda de error pesimista y poda por costos.

4.4.5 Arboles de decisión fuzzy

El proceso de construcción de un árbol fuzzy es similar a la construcción de un árbol de decisión "clásico". [Janikow 1994], [Marsala 2000], [Wang 2004].

La diferencia más importante con respecto a los árboles de decisión "clásicos" radica en la forma que se calcula las frecuencias relativas. En los árboles fuzzy, el grado de pertenencia de un ejemplo con un nodo es un número real en el rango [0,1]. Este número indica el grado de satisfacción de un ejemplo con las restricciones fuzzy que llegan al nodo.

Un ejemplo puede pertenecer a varias hojas, la diferencia va a ser el grado de pertenencia que tiene con cada hoja. El conflicto producido sucede por el solapamiento existente en la definición de los términos lingüísticos de cada variable lingüística. Esta característica de los árboles fuzzy tiene ventajas y desventajas. Las ventajas son: conducta más gradual en el proceso de inferencia y la utilización como manera de enfrentar los problemas de información ruidosa o incompleta. La desventaja es la mayor complejidad en el proceso de inferencia provocada por la inconsistencia. Para lidiar con este problema debemos recurrir al razonamiento aproximado.

Las restricciones fuzzy están formadas por una variable lingüística, un término lingüístico y el operador de igualdad.

Dado que las restricciones fuzzy se evalúan utilizando funciones de membresía, el proceso brinda una manera de relacionar el valor de los dominios continuos con un concepto más abstracto, tales como términos fuzzy. Los datos numéricos, luego de la transformación a términos fuzzy, no son esenciales para el algoritmo porque las métricas se aplican a términos fuzzy. Los dominios inherentemente simbólicos pueden tratarse en el árbol de decisión fuzzy directamente. Cuando se procesan datos simbólicos y se utiliza el algoritmo de árbol de decisión fuzzy, se obtiene el mismo resultado que utilizando el algoritmo de árbol de decisión clásico.

Los árboles se pueden interpretar como base de reglas, por lo tanto, los árboles de decisión fuzzy pueden ser vistos como un medio para aprender reglas fuzzy.

El camino que conduce a cada hoja puede interpretarse como una regla fuzzy. Cada hoja puede tener asociada varias clases, por lo tanto se obtiene un conjunto de reglas fuzzy potencialmente conflictivas.

Se puede extender la noción de proposición clásica a proposición fuzzy de la forma "V = A" donde V es una variable lingüística y A es un valor lingüístico (término fuzzy). Una regla clásica se puede extender a una regla fuzzy, por ejemplo:

Si V1 = A1 **y** V2 = A2 **Entonces** D = D1: Siendo V1 y V2 variables lingüísticas; A1 y A2 son valores lingüísticos o términos fuzzy; (V1 = A1) y (V2 = A2) son restricciones fuzzy.

4.5 Reglas de asociación

El algoritmo de reglas de asociación tiene ciertas similitudes con el algoritmo de árbol de decisión. Ambos algoritmos tienen como objetivo la creación de reglas. Sin embargo, el algoritmo de reglas de asociación fue creado con fines descriptivos y el algoritmo de árbol de decisión con fines predictivos. No obstante, algunos autores utilizan el algoritmo de reglas de asociación con fines predictivos. Esto lo consiguen imponiendo ciertas restricciones sintácticas a las reglas generadas [Au 2001], [Zaiane 2002], [Wong 2000]. Por ejemplo, el atributo clase debe estar siempre presente y ser consecuente de las reglas generadas.

El concepto de reglas de asociación fue inventado por [Agrawal 1993]. Esta tarea de data mining surgió para solucionar el problema del "carrito de supermercado". Las decisiones típicas que toma un gerente de supermercado son, por ejemplo, qué productos poner a la venta, cómo poner la mercadería en los estantes para maximizar la ganancia, etc. Para mejorar la calidad de tales decisiones se recurre generalmente a la historia. Para convertir este volumen gigantesco de datos en información útil para la toma de decisión, se necesitan algoritmos con importante énfasis en la eficiencia.

Un ejemplo del tipo de regla de asociación que se busca en este caso es la siguiente: el 90 % de las compras que incluyen pan y manteca, también incluyen leche. El antecedente de esta regla consiste en los productos pan y manteca, y el consecuente es el producto leche. El número 90 % es el factor de confianza de la regla.

Mediante estas reglas se pueden buscar aquellas que tengan un determinado producto en el antecedente o en el consecuente. Se podría determinar las mejores K reglas que tengan un producto determinado en el consecuente. El término "mejor" regla significa filtrar, por alguna medida, las más significativas. Una de las medidas podría ser la confianza de la regla.

La cantidad de reglas generadas es potencialmente enorme, por lo tanto, estamos interesados en generar sólo las reglas que satisfacen ciertos criterios adicionales.

Los criterios adicionales pueden agruparse en dos categorías:

- Restricciones sintácticas: estas restricciones establecen los ítems que pueden aparecer en una regla. También se puede restringir el lugar donde debe aparecer los ítems, en el antecedente y/o el consecuente, la cantidad de elementos en el antecedente y/o la cantidad de elementos en el consecuente.
- Restricciones de soporte: estas restricciones se refieren a la cantidad de operaciones que "soportan" la regla. El soporte de una regla se define como la fracción de transacciones que contienen los productos del antecedente y los del consecuente. El soporte de una regla no se debería confundir con la confianza de la misma. La confianza es una medida de la fortaleza de la regla, en cambio el soporte corresponde a la significación estadística. Además de la significación estadística, otra motivación para tener en cuenta el soporte es por razones impuesta del negocio. Por ejemplo, si el soporte no es lo suficientemente grande, significa que la regla no se debe considerar porque no vale la pena. Tal vez sean situaciones anómalas que no sirven para tomar una decisión a nivel general. En caso que se utilice la información para detectar anomalías, el interés se pondrá en las reglas con bajo soporte.

Resumiendo, las medidas más importantes que evalúan la calidad de una regla son las siguientes:

Sea una regla con el formato:

$A \rightarrow C$, donde A es el antecedente y C es el consecuente de la regla. A y C son conjuntos de ítems.

Soporte ($A \rightarrow C$) = (cantidad de transacciones que incluyen los ítems presentes en A y C) / (cantidad total de transacciones).

Confianza ($A \rightarrow C$) = soporte ($A \rightarrow C$) / soporte (A).

Soporte (A) = (cantidad de transacciones que tienen los elementos del conjunto A) / (cantidad total de transacciones).

El problema de la búsqueda de reglas de asociación se resume en los siguientes dos pasos:

- Generar las combinaciones de ítems que tengan un soporte mayor a un cierto valor ingresado como parámetro al algoritmo. Las combinaciones cuyo soporte es mayor al mínimo soporte se denominan ítems frecuentes (large itemsets). En este paso se pueden añadir restricciones sintácticas.
- Tomar los large itemsets Y = i_1, i_2, i_3, i_k con k >= 2. Generar todas las reglas que usen los items del conjunto i_1, i_2, i_3, i_k. El antecedente de cada una de estas reglas será un subconjunto X de Y, de tal modo que X tendrá k-1 items. El consecuente de cada regla será Y – X. Para generar una regla interesante, la misma debe satisfacer el factor de confianza.

Habiendo obtenido los large itemsests y su soporte, la solución al segundo paso es fácil. La investigación en los algoritmos para descubrimiento de reglas de asociación centra su interés en el primer paso. Desde el punto de vista computacional, implica desafíos vinculados al balance entre tiempo de ejecución, espacio de almacenamiento disponible y complejidad algorítmica de la solución.

El modelo que vimos es aplicable cuando los atributos son nominales o a lo sumo finito numerables. Por ejemplo, los productos que se compran empaquetados en un supermercado. Esto no es sorprendente ya que el algoritmo se diseño teniendo como objetivo el estudio de las compras en un supermercado.

Cuando se quiere aplicar el algoritmo en dominios con atributos cuantitativos (continuos) y categóricos [Srikant 1996], surge el problema del tratamiento (preparación) de los atributos cuantitativos. Este problema se llama reglas de asociación cuantitativas. Un ejemplo de regla de este tipo es la siguiente:

(Edad: 30…39) y (Casado: sí) → (CantAutos: 2). Donde Edad es un atributo numérico continuo, CantAutos es un atributo numérico discreto y Casado es un atributo nominal.

La solución más directa para resolver este tipo de problema es partir los atributos numéricos en rangos de valores. Esta solución crea dos problemas:

- Mínimo soporte: si el número de intervalos es grande, el soporte para cada simple intervalo será bajo. Esto podría ocasionar que pocos intervalos pasen el criterio del mínimo soporte (valor ingresado como parámetro al algoritmo). Este es el primer paso de los algoritmos de regla de asociación, por lo tanto, si se encuentran pocos itmsets frecuentes, la cantidad de reglas obtenida será muy baja.
- Pérdida de información: siempre que se parta el dominio del atributo en intervalos, se pierde información.

Los problemas planteados generan una situación de compromiso, si los intervalos son demasiados grandes, muchas reglas no pasarán la mínima confianza; en cambio, si son demasiado chicos, muchas reglas no pasarán el mínimo soporte [Megiddo 1998], [Bayardo 1999]. Podríamos disminuir el tiempo de ejecución con menos intervalos y reducir la pérdida de información con más intervalos. Por el contrario, podríamos reducir la pérdida de información incrementando el número de intervalos, a costa de

incrementar el tiempo de ejecución y potencialmente generar muchas reglas poco interesantes.

La técnica de agrupar valores no se aplica a los atributos nominales, ya que es poco significativo agrupar valores nominales, a menos que exista una jerarquía que los agrupe en forma lógica. Usar una taxonomía de esta manera es similar a considerar rangos en atributos cuantitativos [Liu 2000], [Schiaffino 2003].

4.5.1 Reglas de asociación fuzzy

En el apartado anterior vimos que los algoritmos de reglas de asociación solucionan el problema de los atributos cuantitativos, particionando el dominio del atributo, convirtiéndolo en una forma que pueda tratar el algoritmo tradicional. Aunque este método soluciona el problema, causa problemas con los valores que están cerca de los límites de cada intervalo. Utilizando límites definidos ignoramos o enfatizamos la importancia de los valores cercanos a los límites del intervalo [Kuok 1996].

Los métodos de particionamiento citados anteriormente están sujetos a los efectos de los límites definidos con precisión. El problema subyacente de estos métodos es el uso de la teoría clásica de conjuntos.

Para solucionar el problema de los intervalos definidos con precisión, se puede usar lógica difusa. El concepto de fuzzy set es mejor que el método de la partición porque los fuzzy sets brindan una transición más gradual. Debido a la transición más suave, habrá pocos elementos de frontera excluidos. Además, las reglas de asociación fuzzy son más entendibles para el ser humano por el uso de variables lingüísticas y valores lingüísticos. Cada variable lingüística tiene valores lingüísticos asociados, el significado de cada uno de ellos está dado por los fuzzy sets definidos en su respectiva función de membresía.

En la teoría de fuzzy sets los elementos pueden pertenecer a un conjunto con un grado de membresía en el rango [0,1]. Este valor se asigna mediante la función de membresía asociada a cada fuzzy set. De esta forma, el problema de los límites definidos con precisión se soluciona.

Luego de especificar el significado de cada valor lingüístico estamos en condiciones de utilizar el algoritmo de reglas de asociación tradicional. La diferencia más notable entre la versión clásica y fuzzy, esta dada por el cálculo de las métricas: soporte y confianza.

Algunos autores plantean métricas distintas a las tradicionales (soporte y confianza) para la extracción de reglas de asociación con fines predictivos, por ejemplo [Au 2001].

Soporte en reglas fuzzy

Al igual que en las reglas de asociación tradicional, para generar las reglas de asociación fuzzy, se debe encontrar primero el conjunto de ítems frecuentes. Estos últimos deben tener un nivel de significación superior a un valor ingresado como parámetro.

El nivel de significación (soporte) se calcula sumando las contribuciones de cada registro con respecto a un itemset especificado, y luego dividiendo el valor hallado por la cantidad de registros total. Cada registro contribuye con un peso que varía en el rango [0,1].

Para ilustrar el cálculo del soporte de una regla fuzzy recurrimos a un ejemplo. Supongamos que tenemos el siguiente conjunto de registros, en cada uno de ellos se indica el grado de membresía de cada valor lingüístico. El operador de conjunción es el Tnorm multiplicación algebraica.

(Sueldo, alto)	(Balance, bajo)		Mult.
0,9	0,2	R1	0,18
0,2	0,7	R2	0,14
0,5	0,4	R3	0,2
0,3	0,7	R4	0,21
0,6	0,3	R5	0,18

Si Sueldo es Alto **Entonces** Balance es Bajo.

Soporte (Sueldo = ”Alto”, Balance = “Bajo”) = (0.18+0.14+0.2+0.21+0.18) / 5 = 0.182.

El cálculo del soporte de la regla fuzzy se diferencia del soporte de una regla tradicional en la contribución de cada ejemplo. Utilizando lógica tradicional cada ejemplo contribuye con un peso {0,1}.

Confianza en reglas fuzzy

La confianza de una regla fuzzy se obtiene de la misma forma que en la versión clásica.

Confianza (A → C) = soporte (A → C) / soporte (A)

Supongamos el mismo conjunto de ejemplos y regla del ejemplo anterior.

Si Sueldo es alto **Entonces** Balance es bajo.

Soporte (A → C) = 0.182
Soporte (A) = (0.9+0.2+0.5+0.3+0.6) / 5 = 0.5
Confianza (A → C) = 0.364

4.6 Análisis de la base de reglas

En las secciones anteriores vimos dos clases de algoritmos que generan una base de reglas difusas. El algoritmo árbol de decisión esta diseñado para fines predictivos (clasificación), en cambio el algoritmo de reglas de asociación debe ser modificado, ya que fue pensado principalmente para fines descriptivos.

Los algoritmos vistos tienen estrategias distintas para conseguir la base de reglas fuzzy, sin embargo las métricas que caracterizan a cada regla son inherentes a la misma.

Por ejemplo, se podría obtener el soporte y confianza de una regla utilizando el algoritmo árbol de decisión. Esto permite separar la estrategia de hallazgo con el resultado (base de reglas). Por supuesto, cada estrategia puede generar un conjunto de reglas diferentes.

El antecedente de una regla forma una partición fuzzy tipo grilla. Cada celda de la grilla se especifica por la combinación de los valores lingüísticos de cada atributo que figure en el antecedente. La cantidad total de celdas surge de la multiplicación de la cantidad de valores lingüísticos de los atributos.

El siguiente gráfico (*figura 4.5*) muestra la diferencia entre una base de reglas construida con partición difusa y partición fija.

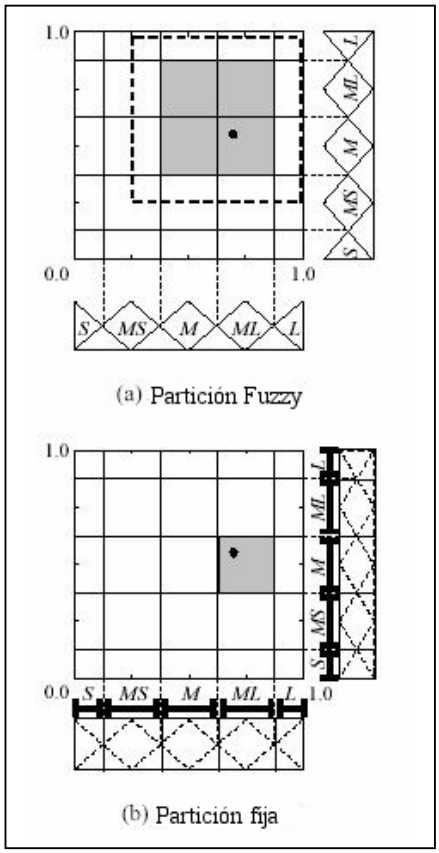


Figura 4.5 Comparación partición fija y partición difusa (fuzzy).

La diferencia más notable es la ausencia de solapamiento entre los intervalos en el caso de partición fija [Nakashima 2000]. Esto hace que se active sólo una regla para cada ejemplo de entrenamiento. En cambio se activan varias reglas cuando la partición es fuzzy.

El gráfico (*figura 4.6*) muestra la ausencia de reglas en una porción del espacio. Esto se debe a la ausencia de ejemplos de entrenamiento en la zona sombreada. Cuando se utiliza partición fuzzy se cubre todo el espacio formado por la conjunción de los atributos.

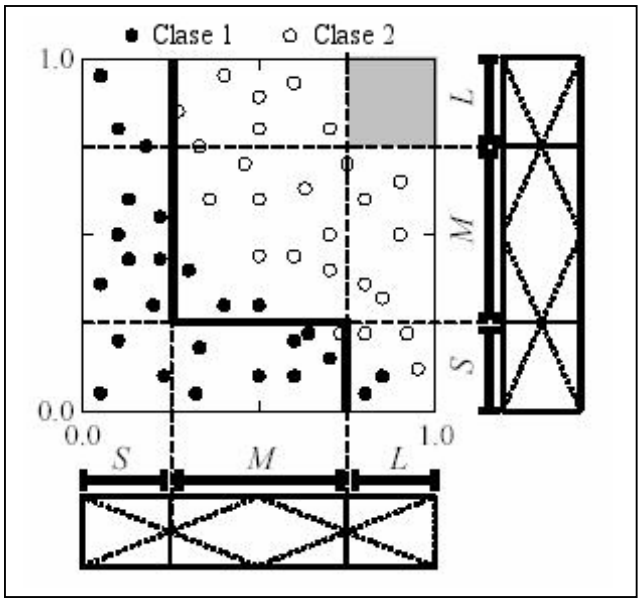


Figura 4.6 Comparación espacio cubierto por la partición fuzzy y la partición fija.

El análisis de los gráficos indica que las particiones fuzzy aumentan la cobertura de la base de reglas construida. Sin embargo, la activación de varias reglas que se produce con un sólo ejemplo de aprendizaje, aumenta la inconsistencia.

El proceso de inferencia general en una base de reglas fuzzy es el siguiente:

- Evaluar el grado de cumplimiento de cada regla con el ejemplo a clasificar.
- Establecer la adhesión a cada clase utilizando el resultado del paso anterior.
- La clase inferida será la que sume mayor adhesión.

Evaluar el grado de cumplimiento de cada regla con el ejemplo a clasificar

Este proceso consiste en los siguientes pasos:

- Evaluar el cumplimiento de cada restricción fuzzy presente en el antecedente de la regla. Esto se realiza utilizando la función de membresía definida para la variable lingüística y valor lingüístico presente en la restricción.
- Evaluar la conjunción (suponemos que las restricciones fuzzy siempre están unidas por el operador de conjunción) de restricciones fuzzy del antecedente, para este propósito se utiliza un operador de conjunción (un t-norm).
- Evaluar la conjunción entre el antecedente y el consecuente de la regla, para este propósito se utiliza un t-norm. En este paso se utiliza el grado de cumplimiento del antecedente, obtenido en el paso anterior, y el factor de certeza de la regla.

Establecer la adhesión a cada clase

- Agrupar las reglas por la clase expresada en el consecuente.
- En cada agrupamiento calcular el grado de adhesión a cada clase utilizando un operador de disyunción (t-conorm).

4.6.1 Cálculos del factor de certeza

La especificación del factor de certeza de una regla tiene un efecto muy importante en el proceso de inferencia. La confianza es la medida más ampliamente usada como factor de certeza de una regla fuzzy. Algunos autores plantean medidas alternativas a la señalada anteriormente [Ishibuchi 2004]. A continuación se describen algunas de ellas, junto con las implicancias que tiene cada una.

Confianza de la regla: en este método el factor de certeza de la regla es directamente la confianza de la misma. Este es el método más sencillo, dado que no habría que hacer ningún cálculo extra. Recordemos que el consecuente de una regla es la clase con mayor confianza. El problema que tiene este método es que no tiene en cuenta los ejemplos que contradicen la regla. Este hecho puede conducir a tener gran cantidad de reglas cuyos consecuentes tengan bajo grado de dominancia. La fórmula sería la siguiente FC = Confianza(A → Clase C).

Grado de dominancia sobre el promedio: en este método se resta a la confianza del consecuente, la confianza promedio de los ejemplos que contradicen la regla. La fórmula es la siguiente: FC = Confianza(A → Clase C) - Confianza(A → Clase <> C)/ (M-1), donde M es la cantidad de clases.

Diferencia con segunda: en este método se resta a la confianza de la clase dominante (el consecuente de la regla), la confianza de la segunda clase dominante.

Diferencia de dominancia: en este método se resta a la confianza del consecuente, la sumatoria de las confianza de las otras clases. Cuando se aplica este método es altamente probable que el factor de certeza resultante sea negativo. Dado que un factor de certeza negativo es inaceptable, se debe eliminar la regla. El problema es opuesto al método de la confianza de la regla; se eliminaría mucha cantidad de reglas, quedarían solamente aquellas reglas con grado de dominancia muy grande.

4.6.2 Ejemplos positivos y negativos

Dado una regla del tipo A → B, en la cual A es el antecedente de la regla y B es el consecuente de la misma y un conjunto de ejemplos X, en el cual cada ejemplo lo denominaremos x, se aplican las siguientes definiciones [Dubois 2003].

- Ejemplo positivo: un ejemplo es positivo si cumple la condición: $x \in A$ y $x \in B$. Este tipo de ejemplo "refuerza" la regla. Si dividimos la cantidad de ejemplos que cumplen la condición por la cantidad total de ejemplos obtenemos el soporte. Tanto el soporte como la confianza "tradicional" de una regla están definidos en base a los ejemplos positivos.

- Ejemplo negativo: un ejemplo es negativo si cumple la condición: $x \in A$ y $x \notin B$. Este tipo de ejemplo contradice la regla, por lo tanto le quita fortaleza.

Los ejemplos no positivos y no negativos ($x \notin A$) no refuerzan ni niegan la regla, son irrelevantes. Por lo tanto el interés para desarrollar medidas de calidad de una regla se enfoca en los ejemplos positivos y negativos.

Capítulo V Algoritmo

5.1 Introducción

En los capítulos anteriores se definió el problema, se comentaron las soluciones alternativas, se presentaron y criticaron los trabajos existentes relacionados con aplicaciones sensibles al contexto. También se bosquejó la solución presente en esta tesis y su conveniencia respecto a los anteriores trabajos.

Para fomentar la modularidad del sistema se dividió el sistema en sus partes constitutivas, en este capítulo vamos a concentrarnos en los algoritmos que se utilizarán en cada una de ellas, los problemas a enfrentar y las alternativas de solución.

5.2 Aprendizaje

Los algoritmos basados en árboles de decisión representan una de las mejores metodologías para adquirir conocimiento de un dominio. La popularidad de los árboles de decisión se debe a sus ventajas, tales como: velocidad de construcción y facilidad de trasladar el conocimiento a un conjunto de reglas accesibles al ser humano [Borisov 2003].

En algunas aplicaciones, los datos que se usan son de naturaleza imprecisa y/o subjetiva. El enfoque más utilizado para capturar esta vaguedad en la información es usar lógica difusa. Los algoritmos de árboles de decisión fuzzy combinan las ventajas de la lógica difusa y los algoritmos de árboles de decisión. La mayoría de estos algoritmos difieren en el criterio (métrica) empleado para determinar el atributo más importante.

Los algoritmos que sólo construyen un árbol de decisión fuzzy desde el principio, sufren de un inconveniente importante, que es su bajo rendimiento cuando deben trabajar en tiempo real. Estos algoritmos no permiten adaptar el resultado a los nuevos datos recibidos. Las aplicaciones sensibles al contexto tienen precisamente esa característica. Para solucionar este problema se modificaron los algoritmos existentes, transformándolos en incrementales. Esta mejora permite la reestructuración del árbol de decisión a medida que se incorporan nuevos datos, lo cual reduce el costo de los cálculos a efectuar. Por lo tanto, el algoritmo que se va a describir tiene las siguientes características:

- Está basado en árboles de decisión
- Utiliza lógica difusa para representar la vaguedad y subjetividad
- Reestructura el conocimiento obtenido a medida que se incorporan nuevos datos

5.2.1 Criterio de selección de atributos

La idea subyacente del proceso de división es obtener particiones “puras”, es decir que la mayoría de los ejemplos pertenezcan a una sola clase. De esta manera, cuando se debe clasificar un ejemplo que pertenece a una partición conocida, se podría inferir con

alto grado de certeza que el ejemplo tiene la misma clase que la mayoría de los ejemplos de la partición. La idea más ampliamente usada para lograr el objetivo mencionado es seleccionar una condición θ en cada paso, tal que las particiones resultantes sean lo más puras posible. Una medida de pureza en la partición λ, es la información dada por los ejemplos incluidos en la misma. La condición θ se forma con el nombre de la variable lingüística y un término lingüístico de dicha variable. Siguiendo el planteo del algoritmo ID3 y para simplificar la lectura de las reglas, las restricciones se forman por igualdad. Por ejemplo: θ = "Si", θ es una variable lingüístico y "Si" es un término lingüístico.

La fórmula sería la siguiente:

$$IC(\lambda) = -\sum_{c \in C} \left[\left(\left| \lambda \cap \mu_c \right| / \left| \lambda \right| \right) * \log\left(\left| \lambda \cap \mu_c \right| / \left| \lambda \right| \right) \right]$$

Siendo:

C = el conjunto de clases posibles.

$IC(\lambda)$ = información contenida en la partición.

$|\lambda|$ = cantidad de ejemplos en la partición (nodo).

$|\lambda \cap \mu_c|$ = cantidad de ejemplos de la clase c en la partición.

Nota: la base del logaritmo es 2.

Para simplificar la exposición y nomenclatura suponemos que tenemos sólo dos clases, y cada variable lingüística tiene asociada dos términos lingüísticos. Dado que el algoritmo en cada paso divide el conjunto de ejemplos en dos particiones con distinto tamaño, podemos estimar la calidad de la división hecha por θ, calculando el promedio ponderado de la información contenida en ambas particiones. Para obtener el promedio ponderado, se multiplica la información contenida en la partición y el tamaño de la misma.

Dado que cada variable puede tomar dos términos lingüísticos, θ simboliza la restricción θ = "Si" y $\overline{\theta}$ = "No". La fórmula tendría tantos sumandos como términos lingüísticos (fuzzy sets) tenga la variable lingüística.

$$ICW(\lambda, \theta) = \left(\left| \lambda \cap \theta \right| / \left| \lambda \right| \right) * IC(\lambda \cap \theta) + \left(\left| \lambda \cap \overline{\theta} \right| / \left| \lambda \right| \right) * IC\left(\lambda \cap \overline{\theta} \right)$$

Siendo:

$ICW(\lambda, \theta)$ = información promedio contenida en la partición hecha por la restricción θ.

$|\lambda \cap \theta|$ = cantidad de ejemplos en la partición dada por el valor θ.

$\left| \lambda \cap \overline{\theta} \right|$ = cantidad de ejemplos en la partición dada por el valor $\overline{\theta}$.

$IC\left(\lambda \cap \overline{\theta} \right)$ = información contenida en la partición dada por el valor $\overline{\theta}$.

$IC(\lambda \cap \theta)$ = información contenida en la partición dada por el valor θ.

La idea básica de la heurística es elegir la condición θ para la partición λ, que tenga el menor contenido de información promedio ICW(λ, θ).

5.2.2 Criterios de parada del algoritmo

El algoritmo de construcción de un árbol de decisión divide el conjunto de ejemplos en forma recursiva utilizando la heurística elegida, en este caso la información promedio. Como en todo algoritmo recursivo se necesita una condición de parada. Los criterios de parada que se usarán son los siguientes:

- La partición tiene una clase claramente dominante. Este criterio significa elegir una frecuencia relativa mínima para considerar una clase dominante. Esta debería ser la condición de parada que más se cumple. Si esta condición se cumple, estaríamos cumpliendo con el objetivo de lograr particiones "puras".

- La partición tiene un tamaño despreciable. Esto significa que la cantidad de ejemplos es muy pequeña, por lo tanto no vale la pena seguir subdividiendo, las subdivisiones que se obtendrían no serían representativas. Esta condición de parada ataca el problema del sobreajuste, problema muy común en los árboles de decisión. Debería tomarse como la segunda condición de parada. Recordemos que en el caso de los árboles de decisión fuzzy, la cantidad de ejemplos en una partición es un número real, dado que cada ejemplo contribuye con un valor de membresía en el rango [0,1].

- No hay atributos disponibles para seguir subdividiendo. Este criterio surge de la restricción de usar a lo sumo una vez un atributo en cada camino del árbol. Esta restricción hace más legible los caminos creados por el árbol. Se debería tomar como el último recurso. Cuando esta condición se dispara significa generalmente problemas en las decisiones tomadas "más arriba" (más cerca de la raíz) en el árbol.

Las condiciones de parada se deben fijar antes de la creación del árbol y permanecerán toda la vida del mismo. Un cambio en estos parámetros significaría la necesidad de construir el árbol desde el principio.

5.2.3 Creación del árbol

El algoritmo de creación del árbol es una modificación del ID3, el procedimiento se adapta para que pueda usar lógica difusa [Guetova 2002].

Algoritmo de creación (pseudocódigo)

Parámetros: un conjunto de ejemplos fuzzy λ, un conjunto de condiciones τ (atributos de inferencia).

Parámetros implícitos: la medida de selección de atributos, el conjunto de criterios de parada.

Salida: un árbol de decisión fuzzy.

Procedimiento

Criterio de parada se cumple?

SI: Retornar el nodo hoja que contiene los ejemplos en la partición.

NO: Buscar la mejor condición θ desde el conjunto de condiciones τ, según la heurística ingresada.

Crear los nodos hijos según el atributo ganador. Se creará un nodo por cada valor lingüístico que tenga dicho atributo.

Retornar el árbol resultante.

5.2.4 Reestructuración del árbol

Cuando arriban nuevos ejemplos de aprendizaje, se tendrá un nuevo árbol resultante de la incorporación de dichos ejemplos. Comparando el árbol viejo y el nuevo, dos cosas podrían cambiar: la distribución de clases en los nodos hoja o la estructura del árbol.

El cambio en la estructura del árbol es el cambio más costoso. Por lo tanto, lo primero que se trata de hacer es incorporar los ejemplos cambiando la distribución de clases en las hojas, la opción menos costosa en términos de cálculo. El único cambio estructural de bajo costo, es expandir un nodo hoja cuando los criterios de parada no se cumplan (por la incorporación de los nuevos ejemplos). Las condiciones en los nodos internos no cambian en este esquema, esto provoca que las mismas no sean necesariamente las sugeridas por la heurística tomada (la medida para seleccionar el atributo ganador). Bajo estas circunstancias, el árbol resultante será diferente del árbol que se construiría si se empezara desde cero. Las pérdidas que podemos tener ejecutando este cambio son: crecimiento inútil del árbol (más nodos que los necesarios) y menor habilidad para clasificar.

El segundo cambio, el cambio estructural, lo debemos hacer frecuentemente para asegurarnos la obtención de un árbol que tenga la misma capacidad de clasificación y tamaño, que si se hubiera hecho desde cero. Esto se consigue corrigiendo las decisiones hechas en los niveles más altos del árbol, teniendo en cuenta el valor de la heurística.

Para cumplir con las demandas citadas anteriormente, el algoritmo de aprendizaje se subdivide en dos partes: el algoritmo de agregado de ejemplos, que haría la actualización menos costosa y el algoritmo de optimización, que haría la actualización más costosa, la reestructuración. El algoritmo de optimización utiliza un procedimiento de transposición de nodos, para lograr que el tope de cada subárbol sea el indicado por la heurística y se realice la mínima cantidad de cálculos posible.

El procedimiento que tomará el algoritmo sería el siguiente:

- Crear el árbol con el primer conjunto de ejemplos.
- Actualizar el árbol cuando arriba un ejemplo con el algoritmo Agregar Ejemplos
- Reestructurar el árbol cuando se hayan ingresado x nuevos ejemplos o cuando el árbol haya crecido más de lo previsto. La variable x es un valor que se ingresaría como parámetro.

La idea de la reestructuración consiste en dos pasos:

- Hallar la mejor condición para subdividir cada nodo.
- Cambiar la estructura del árbol para reflejar el cambio en la heurística.

Tanto el cálculo de la heurística como la reestructuración deben hacerse de forma tal que sea menos oneroso, en términos computacionales, que hacer el cálculo y reestructuración desde cero. Para este propósito debemos reutilizar los cálculos ya efectuados. Esto sugiere almacenar los cálculos hechos en cada nodo para no volver a realizarlos después. Estos valores se deben actualizar cada vez que se agrega un ejemplo o se cambia la estructura.

Para el cálculo de la heurística, en este caso la información promedio, se deben calcular los siguientes valores para todas las clases $c \in C$ y todas las condiciones $\theta \in \tau$.

$|\lambda_t|$ = cantidad de ejemplos total en la partición (nodo t).

$|\lambda_t \cap \mu_c|$ = cantidad de ejemplos de clase $c \in C$, en la partición (nodo t).

$|\lambda_t \cap \theta|$ = cantidad de ejemplos que cumplen la condición θ = "Si" en la partición.

$|\lambda_t \cap \overline{\theta}|$ = cantidad de ejemplos que cumplen la condición θ = "No" ($\overline{\theta}$) en la partición.

$|\lambda_t \cap \mu_c \cap \theta|$ = cantidad de ejemplos que cumplen la condición θ = "Si" y pertenecen a la clase $c \in C$.

$|\lambda_t \cap \mu_c \cap \overline{\theta}|$ = cantidad de ejemplos que cumplen la condición θ = "No" y pertenecen a la clase $c \in C$.

Estos valores los debemos almacenar en los nodos, de tal modo que la heurística se pueda calcular fácilmente cuando lleguen nuevos ejemplos.

Cuando ingresa un nuevo ejemplo, puede darse el caso que no afecte la heurística de todos los nodos. Los nodos que no son afectados no necesitan recálculo de valores, por lo tanto, le haremos una marca sólo a los afectados. De esta manera el algoritmo se torna más eficiente.

5.2.4.1 Agregar ejemplos

El objetivo de este algoritmo es actualizar en forma recursiva los valores almacenados en los nodos. Si los criterios de parada no se cumplen en algún nodo hoja, el nodo se subdivide.

Parámetros: un árbol de decisión fuzzy con un nodo raíz t, un ejemplo e y el conjunto de condiciones τ.
Parámetros implícitos: la heurística de selección y los criterios de parada.
Salida: un árbol de decisión fuzzy t actualizado con el ejemplo e.

Recorrer los nodos del árbol desde la raíz hasta las hojas.
El grado de membresía del ejemplo con la condición del nodo tratado = 0?
  Si: volver
  No: t es un nodo interno?
    Si: Actualizar los valores del nodo t con el ejemplo nuevo.

Ejecutar AgregarEjemplo en los hijos de t.

No: Actualizar los valores del nodo t con el ejemplo nuevo.

El criterio de parada no se cumple más y existen condiciones remanentes?

Si: Elegir la condición ganadora θ.

Reemplazar t (hoja) con un árbol que tenga como raíz a t y como hijos los valores lingüísticos de la condición ganadora.

Agregar los ejemplos del nodo t a los hijos de éste, de a uno por vez.

5.2.4.2 Optimizar árbol

Este algoritmo tiene dos objetivos:

- Asegurar que el nodo raíz de cada subárbol tenga asociada la subdivisión que determina la heurística.
- Asegurar que las condiciones de parada se cumplan sólo en las hojas.

Para lograr los objetivos señalados, este algoritmo llama a Transponer árbol cuando se necesita cambiar la estructura sin cambiar la cantidad de ejemplos de los nodos hijos. Básicamente, las restricciones necesarias para llegar a cada nodo se mantienen, sólo cambia el orden de las mismas.

Optimizar

Parámetros: un árbol de decisión fuzzy (t) y las condiciones posibles.
Parámetros implícitos: la heurística de selección y los criterios de parada.
Salida: un árbol de decisión fuzzy, en el cual las condiciones en cada subárbol es la indicada por la heurística y los criterios de parada se cumplen sólo en las hojas.

T es hoja o condiciones posibles = vacío?
Si: devuelve t.
T cumple los criterios de parada?
Si: devuelve t como hoja (poda el árbol).
Nodo t está marcado como "sin actualizar"?
Si: buscar la condición indicada por la heurística.
Asignar nodo t := Transponer árbol(t, condición heurística, condiciones posibles).
Poner marca de "actualizado".
Reemplazar los hijos de t llamando en forma recursiva a Optimizar

Transponer árbol

El objetivo de este algoritmo es forzar que la condición en la raíz del árbol sea la indicada por el parámetro. El proceso se debe hacer sin cambiar la cantidad de ejemplos en las hojas, ni las condiciones acumuladas para llegar a cada nodo. En caso contrario el proceso sería equivalente en esfuerzo a crear el árbol nuevamente.

El algoritmo tiene dos partes: la primera consiste en forzar la creación de la condición determinada por la heurística (en caso que sea diferente a la anterior), y la

segunda es cambiar la estructura, de manera que se mantengan las condiciones para llegar a las hojas.

El tamaño de la partición en las hojas (cantidad de ejemplos) no cambia por la corrida del algoritmo, ni tampoco lo hace la raíz del árbol mayor, dado que no se agregan ejemplos reales al árbol.

Los conteos almacenados en los nodos de nivel intermedio cambian porque tienen nodos hijos distintos a los que tenían antes de la corrida del algoritmo. Los valores almacenados en los nodos, junto al operador de conjunción (T-norm) entre restricciones fuzzy elegido (la multiplicación del grado de membresía), brindan la posibilidad de calcular la heurística de selección sin recurrir a la base de ejemplos.

Transponer

Parámetros: un árbol de decisión fuzzy t, una condición a instalar en el tope de la estructura y las condiciones remanentes.

Parámetros implícitos: la heurística de selección y los criterios de parada.

Salida: un árbol de decisión con la condición indicada como parámetro.

T es un nodo interno con la condición de parámetro?
  Si: devuelve el nodo t
Repetir por cada hijo t' del nodo t
  T' es una hoja?
    Si: reemplazar el nodo hoja por un árbol con la condición de parámetro y agregarle los ejemplos con el algoritmo Agregar ejemplos.
    No: t':= Transponer(t', decisión de parámetro, condiciones remanentes)
Cambiar los nodos internos de tal manera que la decisión requerida esté en el tope de la estructura, y la que estaba en el tope baje de nivel.
Recalcular los conteos almacenados de los nodos internos que cambiaron sus hijos
Poner la marca de "sin actualizar" a los nodos recalculados.

Para mejorar la perfomance del algoritmo optimizador (recorra una estructura más chica), se debe eliminar las hojas con tamaño de partición cero (éstas surgen del cambio de estructura) y reemplazar el padre por el nodo hermano de la hoja eliminada (suponiendo que el hermano contiene todos los ejemplos).

Ejemplo de cálculos

En un sitio web se desea adaptar el contenido de la página mostrada a un usuario, según el interés que manifiesta el mismo en cada tópico. El interés del usuario en un tópico se demuestra cuando agranda o achica las ventanas que se le presentan. Se supone que cada ventana que se le presenta al usuario tiene un componente técnico, componente de diseño y un componente de imagen conocido. El problema es predecir (clasificar) la respuesta que tendrá el usuario cuando se le presenta información (página web) con componentes técnico, diseño e imagen conocido. La predicción se realiza teniendo en cuenta la historia.

Clases Respuesta (R) [Positiva, Negativa].

Atributos (variables fuzzy): Componente Técnico (T), Componente Diseño (D), Componente Imagen (I).

Cada variable tiene asociados dos términos fuzzy: Sí y No. El grado de membresía representa la componente de cada atributo que tiene la página.

Los ejemplos de aprendizaje representan páginas mostradas al usuario, es decir, interacciones con el mismo.

El siguiente ejemplo muestra la primera iteración del algoritmo. Sean los siguientes criterios de parada y conjunto de ejemplos.

Pureza mínima partición = 90%
Tamaño mínimo relevante = 1.5 ejemplos

Ejemplo	Imagen		Técnico		Diseño		Respuesta	
1	(Si;1)	(No;0)	(Si;0,5)	(No;0,5)	(Si;0,4)	(No;0,6)	(Pos;0)	(Neg;1)
2	(Si;0)	(No;1)	(Si;0,6)	(No;0,4)	(Si;0,6)	(No;0,4)	(Pos;0)	(Neg;1)
3	(Si;0)	(No;1)	(Si;0,8)	(No;0,2)	(Si;0,5)	(No;0,5)	(Pos;1)	(Neg;0)

Las pureza en el nodo raíz (partición inicial) es 66,66% (menor al parámetro 90%), por lo tanto se debe encontrar el mejor atributo para expandir el árbol y conseguir mayor pureza.

Imagen

θ: Imagen = Si
$|\lambda \cap \theta|$: 1
$|\lambda \cap \mu_c \cap \theta|$: 0, para clase Pos
$|\lambda \cap \mu_c \cap \theta|$: 1, para clase Neg

$IC(\lambda \cap \theta)$ = -[((1/1)*log(1/1))+((0/1)*log(0/1))] = 0

$\overline{\theta}$: Imagen = No
$|\lambda \cap \overline{\theta}|$: 2
$|\lambda \cap \mu_c \cap \overline{\theta}|$: 1, para clase Pos
$|\lambda \cap \mu_c \cap \overline{\theta}|$: 1, para clase Neg

$IC(\lambda \cap \overline{\theta})$ = -[((1/2)*log(1/2))+((1/2)*log(1/2))] = 1

Técnico

θ: Técnico = Si
$|\lambda \cap \theta|$: 1.9
$|\lambda \cap \mu_c \cap \theta|$: 0.8, para clase Pos
$|\lambda \cap \mu_c \cap \theta|$: 1.1., para clase Neg

$IC(\lambda \cap \theta)$ = -[((1.1/1.9)*log(1.1/1.9))+((0.8/1.9)*log(0.8/1.9))] = 0.98

$\overline{\theta}$: Técnico = No

$|\lambda \cap \overline{\theta}|$: 1.1

$|\lambda \cap \mu_c \cap \overline{\theta}|$: 0.2, para clase Pos

$|\lambda \cap \mu_c \cap \overline{\theta}|$: 0.9, para clase Neg

$IC(\lambda \cap \overline{\theta})$ = -[((0.9/1.1)*log(0.9/1.1))+((0.2/1.1)*log(0.2/1.1))] = 0.68

Diseño

θ: Diseño = Si

$|\lambda \cap \theta|$: 1.5

$|\lambda \cap \mu_c \cap \theta|$: 0.5, para clase Pos

$|\lambda \cap \mu_c \cap \theta|$: 1, para clase Neg

$IC(\lambda \cap \theta)$ = -[((1/1.5)*log(1/1.5))+((0.5/1.5)*log(0.5/1.5))] = 0.92

$\overline{\theta}$: Diseño = No

$|\lambda \cap \overline{\theta}|$: 1.5

$|\lambda \cap \mu_c \cap \overline{\theta}|$: 0.5, para clase Pos

$|\lambda \cap \mu_c \cap \overline{\theta}|$: 1, para clase Neg

$IC(\lambda \cap \overline{\theta})$ = -[((1/1.5)*log(1/1.5))+((0.5/1.5)*log(0.5/1.5))] = 0.92

<u>Cálculo de ICW</u>

$$ICW\ (\lambda,\theta) = (|\lambda \cap \theta|/|\lambda|) * I_c(\lambda \cap \theta) + \left(|\lambda \cap \overline{\theta}|/|\lambda|\right) * I_c(\lambda \cap \overline{\theta})$$

Imagen

ICW = 1/3 * 0 + 2/3 * 1 = 0.66

Técnico

ICW = 1.9/3 * 0.98 + 1.1/3 * 0.68 = 0.87

Diseño

ICW = 1.5/3 * 0.92 + 1.5/3 * 0.92 = 0.92

El atributo elegido para dividir el nodo raíz es Imagen porque tiene el menor valor de Información promedio.

La división crea dos "ramas": Imagen = Si e Imagen = No.

El nodo con la condición Imagen = Si tiene una pureza del 100%, por lo tanto no se debe dividir.

El nodo con la condición Imagen = No tiene una pureza del 50%, la cantidad de ejemplos es 2, por lo tanto se debe dividir. El conjunto de atributos candidatos para dividir el nodo está integrado por los atributos remanentes (Técnico, Diseño).

5.2.4.3 Fuzzificador

El propósito de este algoritmo es convertir el valor numérico proveniente del sensor y convertirlo en términos lingüísticos. El proceso se efectúa aplicando la función de membresía relacionada con la variable lingüística.

La función de membresía debe establecerse antes de la corrida del algoritmo, generalmente, resulta de la opinión de expertos en el dominio. La elección de la cantidad y amplitud de los términos fuzzy que se definen en la función de membresía, impactan fuertemente en las etapas de aprendizaje, inferencia y la inteligibilidad del conjunto de reglas resultantes. En esta tesis suponemos que la función de membresía está definida para cada atributo.

En los capítulos anteriores vimos que había dos formas generales para representar una función de membresía, ellas son: fuzzy sets triangulares y fuzzy sets trapezoidales. Recordemos además que las funciones de membresía tienen como rango [0,1] y para ser una función de membresía adecuada para el algoritmo, esta debe estar normalizada. Esto último significa que para cada valor del dominio del atributo, la suma de los grados de pertenencia con los fuzzy sets definidos para el mismo, debe ser 1. Por ejemplo, la temperatura 35° C es alta con grado de pertenencia 0.7, es media con grado de pertenencia 0.3 y baja con grado de pertenencia 0. Siendo los valores lingüísticos del atributo {alta, media, baja}.

Para definir un fuzzy set triangular necesitamos almacenar tres valores: dos valores en los cuales la función es 0, y un valor en el cual la función es 1.

Para definir un fuzzy set trapezoidal necesitamos almacenar cuatro valores: dos valores en los cuales la función es 0, y dos valores en los cuales la función es 1.

5.2.5 Poda de reglas

El algoritmo de árbol de decisión tiene el inconveniente de ajustarse demasiado a la muestra con la cual fue creado (sobreajuste). Esto puede ocasionar un bajo desempeño en la etapa de inferencia, en particular cuando se necesita clasificar un ejemplo considerablemente diferente a los ejemplos de aprendizaje.

Para mitigar el efecto señalado se desarrollaron varios métodos de poda, los mismos se pueden clasificar en los siguientes grupos [Orallo 2004]:

- Prepoda: se realizan durante la construcción del árbol.
- Pospoda: el proceso se realiza luego de la construcción inicial del árbol.

Los métodos de pospoda obtienen mejores resultados, pero son menos eficientes que los métodos de prepoda, ya que eliminan nodos de una estructura ya creada. La combinación de ambos métodos es posible y deseable.

La mayoría de los métodos de pospoda utilizan un conjunto de ejemplos de validación diferente a los de aprendizaje, además necesitan parámetros adicionales para determinar la conveniencia de la eliminación de restricciones o reglas. Estos métodos son adecuados cuando el algoritmo se utiliza en forma "Off-line".

Un subtipo de los métodos de pospoda particularmente interesante es el de poda virtual. Esta consiste en reestructurar el árbol utilizando el mismo conjunto de aprendizaje. Este tipo de poda es adecuada cuando la evidencia se obtiene en forma incremental. Por esta razón, la poda virtual (implementada por el algoritmo Optimizar) es el método utilizado en esta tesis.

5.2.6 Métricas de aprendizaje

Antes de describir algunas métricas que caracterizan el proceso de aprendizaje en árboles de decisión, debemos tener en cuenta los siguientes conceptos:

Conjunto de reglas completo: cuando están definidas todas las reglas que se obtienen por las combinaciones posibles de todos los términos fuzzy y todas las variables fuzzy utilizadas para inferir. Por ejemplo, si se tiene tres variables fuzzy de inferencia, y cada una de ellas tiene tres términos fuzzy, se deberían definir nueve reglas que cubran el espacio muestral.

Conjunto de reglas consistente: cuando cada regla definida tiene pureza total (confianza 1).

Desde el punto de vista de la clasificación, obtener un conjunto de reglas que cumpla los conceptos anteriores, sería el ideal. En los procesos inductivos se obtiene un modelo de comportamiento general en base a una muestra. Por lo tanto, el objetivo del proceso es obtener un conjunto de reglas completo respecto de la evidencia suministrada, asumiendo que la muestra es representativa de la población. Cuando se utiliza un algoritmo incremental, la muestra total no se conoce de antemano. Sin embargo, es deseable conocer el ideal para conocer la calidad del aprendizaje.

Cuando se utiliza lógica difusa (tal como se explicó en el capítulo de conceptos básicos), los problemas de inconsistencia aumentan dramáticamente, por lo tanto la obtención de un conjunto de reglas consistente es poco probable.

Por otra parte, se debe tener en cuenta los tópicos que hacen ininteligible al conjunto de reglas. Estos son:

- Cantidad de reglas
- Cantidad de atributos en los antecedentes de las reglas
- Redundancia
- Inconsistencia

En un árbol de decisión, la cantidad de hojas obtenidas representa la cantidad de reglas definidas y la clase dominante (de la hoja) representa el consecuente de la regla.

Las siguientes métricas se pueden utilizar para medir la efectividad del proceso de aprendizaje.

Porcentaje hojas definidas = cantidad hojas creadas/ cantidad hojas posibles.

Porcentaje hojas puras = cantidad hojas puras/ cantidad hojas creadas.

Cantidad hojas puras: es la cantidad de hojas que superan el parámetro de pureza suministrado al algoritmo.

Promedio de restricciones = suma de cantidad de restricciones por hoja/ cantidad de hojas.

Un proceso de aprendizaje (basado en árbol de decisión) es bueno cuando reúne las siguientes condiciones:

- Bajo porcentaje de hojas definidas: significa que el algoritmo discriminó variables y/o términos fuzzy irrelevantes, disminuyendo la cantidad de reglas necesarias para contener los ejemplos de aprendizaje.
- Bajo promedio de restricciones: significa que existe alta proporción de reglas generales, disminuyendo el problema del sobreajuste.
- Alto porcentaje de hojas puras: significa que el algoritmo utilizó frecuentemente la condición de parada de pureza. Por lo tanto, las reglas definidas tienen alto grado confianza.

Los algoritmos incrementales reciben, generalmente, una muestra inicial poco representativa del problema. Al incrementarse la calidad de la muestra, el proceso de aprendizaje "madura", disminuyendo la necesidad de cambios estructurales y mejorando la calidad de las reglas generadas. Las métricas explicadas anteriormente sirven para conocer el grado de maduración del proceso de aprendizaje.

Existen distintas maneras de mejorar las métricas de aprendizaje con la misma muestra, las más importantes son las siguientes:

- Modificar la cantidad de términos fuzzy por variable de inferencia.
- Modificar las funciones de membresía
- Modificar las condiciones de parada (pureza mínima y cantidad mínima de ejemplos relevantes).

Cualquiera de las modificaciones citadas anteriormente, significaría la necesidad de construir el árbol nuevamente, desechando la estructura actual. El único parámetro que puede modificarse dinámicamente es la frecuencia de optimización. Cuando el proceso de aprendizaje está madurando, se requiere optimizaciones más frecuentes que cuando está maduro.

5.3 Inferencia

En este apartado nos concentramos en las maneras de llegar a una inferencia, es decir, encontrarle la clase más apropiada para un ejemplo dado. La decisión la debemos tomar basándonos en la base de reglas fuzzy extraídas en el proceso de aprendizaje.

A continuación vamos a describir dos de los métodos de inferencia más utilizados, la lista completa es muy grande, dado que se pueden hacer pequeñas variaciones a cada uno de ellos [Janikow 1996].

5.3.1 Métodos

Winner rule

El algoritmo de la regla ganadora es sencillo y rápido, por lo tanto, es adecuado para aplicaciones en los cuales el tiempo de respuesta es más importante que la precisión.

La clasificación se logra de la siguiente manera:

- Hallar la compatibilidad del ejemplo a clasificar con cada regla de la base. Este proceso consiste en obtener el grado de compatibilidad con el antecedente de la regla y luego relacionarlo con el factor de certeza de la misma, por medio de un T-norm. El T-norm más utilizado para este propósito es la multiplicación algebraica.

- Elegir la regla que tenga mayor grado de compatibilidad total (calculado en el procedimiento anterior). La clase adjudicada al ejemplo será el consecuente de la regla ganadora.

Average weigth

Este método toma en cuenta el grado de compatibilidad del ejemplo a clasificar con todas las reglas. El consecuente de una regla representa la clase dominante (confianza más alta) de los ejemplos de aprendizaje que cumplen las restricciones del antecedente. Un grado de dominancia menor a 1 indica la existencia de reglas "ocultas" que, teniendo el mismo antecedente, tienen un consecuente distinto a la clase dominante. El presente método tiene en cuenta el grado de compatibilidad del ejemplo con las reglas explícitas y ocultas.

La idea central del método es resumir en un promedio ponderado, el grado de compatibilidad del ejemplo a clasificar con cada clase.

En la hoja de un árbol de decisión se dispone de la información necesaria para el cálculo propuesto por el método. Por esta razón es habitual su utilización cuando la base de reglas deriva de un árbol de decisión.

Las ventajas de este método son: brindar una idea del grado de certeza con el cual se obtiene la clasificación y tener en cuenta la evidencia repartida en todas las reglas. Los grados de pertenencia con cada clase suman 1, por lo tanto, tiene una interpretación similar a una distribución de probabilidad.

Las desventajas del método son: la elevada cantidad de cálculos a realizar y la utilización de reglas "ocultas" para el cálculo.

El procedimiento es el siguiente:

- Hallar la compatibilidad del ejemplo a clasificar, con cada regla (oculta o explícita). Este proceso consiste en obtener el grado de compatibilidad del ejemplo con el antecedente de la regla, y luego relacionarlo con el factor de certeza de la misma por medio de un T-norm. El T-norm más utilizado para este propósito es la multiplicación algebraica.

- Sumar el grado de compatibilidad hallado en la etapa anterior por cada clase. La clase asignada al ejemplo será la clase que tenga mayor suma de grado de compatibilidad.

Ejemplo de inferencia

En base a la estructura aprendida con los ejemplos señalados en la etapa de aprendizaje, se mostrará un ejemplo de clasificación aplicando el método Average Weigth.

En el ejemplo suponemos que los valores de los atributos de inferencia ya fueron fuzzyficados, por lo tanto se indica el grado de membresía con cada término fuzzy definido para cada variable lingüística.

Ejemplo	Imagen		Técnico		Diseño		Respuesta	
4	(Si;0)	(No;1)	(Si;0,3)	(No;0,7)	(Si;0,9)	(No;0,1)	(Pos;?)	(Neg;?)

Base de reglas

Nota: (.) significa confianza de la regla. Las reglas explícitas están resaltadas.

Clase Positivo

Imagen = Si → Pos (0) Comp = 0
Imagen = No AND Técnico = Si → Pos (0,57) Comp = 1*0,3*0,57 = 0,17
Imagen = No AND Técnico = No → Pos (0,33) Comp = 1*0,7*0,33 = 0,23

Grado compatibilidad Pos = 0+0,17+0,23 = 0,4

Clase negativo

Imagen = Si → Neg (1) Comp = 0
Imagen = No AND Técnico = Si → Neg (0,43) Comp = 1*0,3*0,43 = 0,13
Imagen = No AND Técnico = No → Neg (0,67) Comp = 1*0,7*0,67 = 0,47

Grado compatibilidad Pos = 0+0,13+0,47 = 0,6

Valor inferido = Pos

5.3.2 Métricas de inferencia

La evaluación del desempeño de un método de clasificación se puede realizar por medio de un conjunto de métricas. Las medidas definidas a continuación son aplicables a cualquier sistema de clasificación. La definición de las métricas proviene de la documentación del sistema Weka, framework especializado en tareas de data mining [Weka 2004]. Las siguientes medidas se calculan por cada clase, por lo tanto el término "positivo" significa "perteneciente" a la clase que se está midiendo.

PV(positivos verdaderos): es la cantidad de ejemplos clasificados correctamente como pertenecientes a la clase.

PF(positivos falsos): es la cantidad de ejemplos clasificados erróneamente como pertenecientes a la clase.

TPR(total positivos reales): es la cantidad total de ejemplos que realmente pertenecen a la clase.

TPC(total clasificados positivos): es la cantidad total de ejemplos clasificados como positivos.

Precisión: PV/ TPC. La precisión indica la proporción de ejemplos clasificados correctamente como pertenecientes a la clase, sobre la cantidad de ejemplos clasificados como pertenecientes a la clase.

Alcance: PV/TPR. El alcance indica la proporción de ejemplos clasificados correctamente como pertenecientes a la clase, sobre la cantidad de ejemplos realmente pertenecientes a la clase.

Las siguientes medidas se calculan para el modelo (no se refieren a una clase determinada):

PA(porcentaje aciertos): indica el porcentaje de correctos sobre el total de ejemplos.

ER(errores): es la suma de la cantidad de clasificaciones incorrectas y la cantidad de casos no clasificados. Estos últimos se producen cuando el algoritmo no puede asignar ninguna clase.

PE(porcentaje errores): indica el porcentaje de errores sobre el total de casos.

Capítulo VI Arquitectura del sistema

6.1 Introducción

En este capítulo se describirá los subsistemas de software que integran el sistema. La división está pensada para permitir la evolución independiente de cada una de ellas.

El objetivo del sistema es obtener un aspecto de contexto de alto nivel a partir de aspectos de contexto atómicos. Dichos aspectos de contexto atómicos surgen del valor recolectado por medio de sensores.

Los subsistemas que integran el software se describirán en detalle.

6.2 Arquitectura general del sistema

La arquitectura del sistema está formada por cuatro subsistemas, cada uno de ellos tiene una misión específica. El diseño busca separar los conceptos, mejorando de esta manera la evolución independiente de cada subsistema (*figura 6.1*).

En el sistema, cada atributo representa una variable lingüística. Los fuzzy sets definidos para cada atributo representan los valores lingüísticos de la variable lingüística asociada.

Los subsistemas que integran el software son los siguientes:

Control principal: este módulo tiene como misión principal coordinar todos los subsistemas.

Aprendizaje: este módulo tiene como objetivo extraer un modelo que represente los ejemplos de aprendizaje.

Control de sensores: este módulo tiene como objetivo relacionar los sensores con los aspectos de contexto y formar el ejemplo para clasificación.

Repositorio de casos: este módulo tiene como objetivo proporcionar servicios de conteo a los algoritmos de aprendizaje.

Inferencia: este módulo tiene como objetivo inferir el valor del aspecto de contexto de alto nivel.

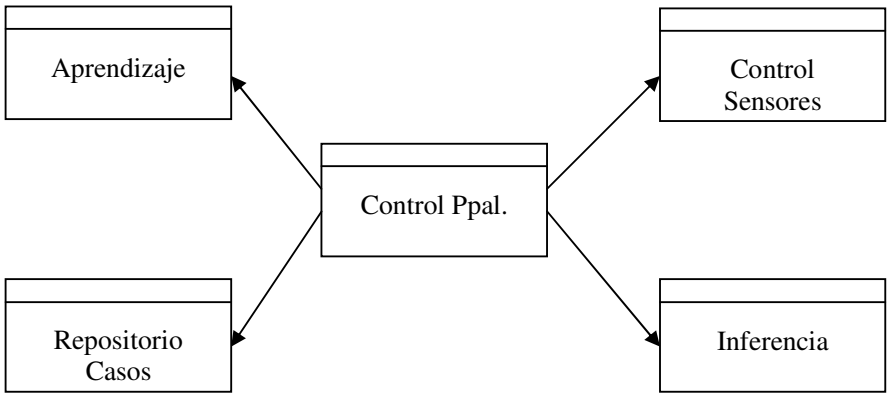


Figura 6.1 Arquitectura general de sistema

6.3 Sensado

El subsistema de sensado consta de dos objetos importantes: el Control de sensores y los sensores. El sensado tiene como objetivo tomar los datos provenientes de sensores y convertirlos en objetos que puedan ser procesados en la inferencia o el aprendizaje.

El control de sensores es el objeto que coordina el proceso de sensado. Los sensores son los objetos que recogen valores directamente del ambiente (contexto). Para cumplir su tarea, el control de sensores almacena una lista de sensores con los cuales debe comunicarse. Además de los sensores que administra, necesita saber la relación que existe entre cada sensor y el aspecto de contexto atómico al cual mide.

Los atributos de inferencia representan los aspectos de contexto atómicos y el atributo clase es el aspecto de contexto derivado no determinístico.

El control de sensores implementa un mecanismo de dependencias para administrar la llegada de nueva evidencia por intermedio de los sensores. Para este propósito cada sensor es "observado" (en los términos de patrones [Gamma 1995], [Alpert 1998]) por el control de sensores, y a su vez el control de sensores es "observado" por el control principal.

El subsistema de sensado está formado por las siguientes clases (*figura 6.2*).

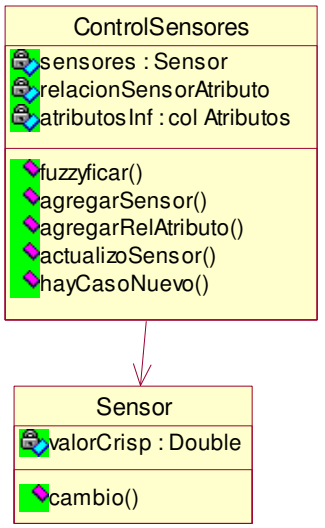


Figura 6.2 Subsistema de sensado

ControlSensores: esta clase tiene como objetivo coordinar la relación entre los sensores y los atributos de inferencia.

Variables de instancia

ControlSensores>sensores: esta variable contiene los sensores conectados al sistema.

ControlSensores>relacionSensorAtributo: esta variable define la relación sensor-atributo de inferencia.

ControlSensores>atributosInf: esta variable contiene los atributos de inferencia que se utilizarán en el proceso de inferencia.

Métodos

ControlSensores>agregarSensor: unSensor
Objetivo: agregar la dependencia del sistema a un sensor.

ControlSensores>agregarRelAtributo: unAtributo sensor: unSensor
Objetivo: ligar cada sensor con el atributo de inferencia correspondiente.

ControlSensores>actualizoSensor: unSensor
Objetivo: avisar al control de sensores del cambio de un sensor.
Pseudocódigo

controlSensores fuzzyficar.

ControlSensores>fuzzyficar
Objetivo: armar el caso de inferencia recorriendo todos los sensores vinculados a los atributos de inferencia.
Pseudocódigo

Por cada atributo inf de inferencia hacer:
  Sensor:= buscar sensor relacionado a atributo inf.
  valorSensor:= sensor valorCrisp.
  valorFuzzy:= inf fuzzyficar: valorSensor.
  Asignar valorFuzzy al atributo inf.
Fin cada atributo.
casoLeido:= ValorFuzzy de todos los atributos de inferencia.

El valor fuzzy de un atributo está formado por el grado de membresía que le corresponde a cada valor lingüístico asociado al atributo (variable lingüística).

ControlSensores>hayCasoNuevo: unCaso
Objetivo: avisar al Control principal la existencia de un nuevo caso para clasificar.
Pseudocódigo

"El control de sensores le avisa al control ppal que tiene un caso nuevo para clasificar"

Observador:= ControlSensores observador (en este caso el Control principal es el observador del Control de sensores).
Observador nuevoCaso: unCaso.

Sensor: esta clase recibe el valor proveniente del ambiente.

Variables de instancia

Sensor>valorCrisp: esta variable guarda el valor recogido por el sensor.

Métodos

Sensor>cambio
Objetivo: implementar la dependencia entre el sensor y el control de sensores.
Pseudocódigo
"El sensor le avisa al control de sensores que cambio su valor"

observador:= sensor observador (en este caso el Control de sensores es el observador del sensor).
observador actualizoSensor: sensor.

6.3.1 Funcionamiento

Cuando un sensor cambia su estado, envía a su "observador", el control de sensores, un mensaje de actualización. El parámetro del mensaje es el mismo sensor, esto permite al control de sensores identificar el sensor que cambió de valor.

Luego de recibir el mensaje de cambio de estado por parte de un sensor, el control de sensores llama al método fuzzificar. El método fuzzificar tiene como objetivo convertir los datos crudos proveniente de sensores en un caso admisible por el algoritmo de inferencia. Para cumplir este propósito toma todos los atributos designados para inferir y busca qué sensor lo mide a cada uno. Una vez que se identifica el sensor relacionado con cada atributo, le pide al atributo que convierta el valor "crisp", proveniente del sensor, en un valor fuzzy.

El caso enviado a inferir estará formado por el valor fuzzy correspondiente a cada atributo de inferencia.

Cuando el control de sensores termina de armar un caso para inferir, se envía un mensaje a sí mismo para avisar que formó el caso. Dado que el control de sensores es "observado" por el control principal, se desencadena un mecanismo de dependencia que finaliza en la clasificación del caso. Este proceso es el nexo entre el mecanismo de sensado y el mecanismo de inferencia, administrado por el control de sensores.

6.4 Aprendizaje

El subsistema de aprendizaje tiene el objetivo de construir un modelo con fines predictivos en base a la evidencia suministrada. El modelo cambia a medida que se va incorporando mayor evidencia. Esto implica que las reglas de comportamiento se modifican durante la ejecución de la aplicación.

El algoritmo de aprendizaje implementado en esta tesis es un árbol de decisión difuso, dado que se lo consideró como el más apto para los objetivos planteados. La arquitectura permite modificar el algoritmo, o más aún, utilizar otro algoritmo de aprendizaje.

El subsistema de aprendizaje está formado por los objetos: AlgoritmoArbol y NodoArbol (*figura 6.3*).

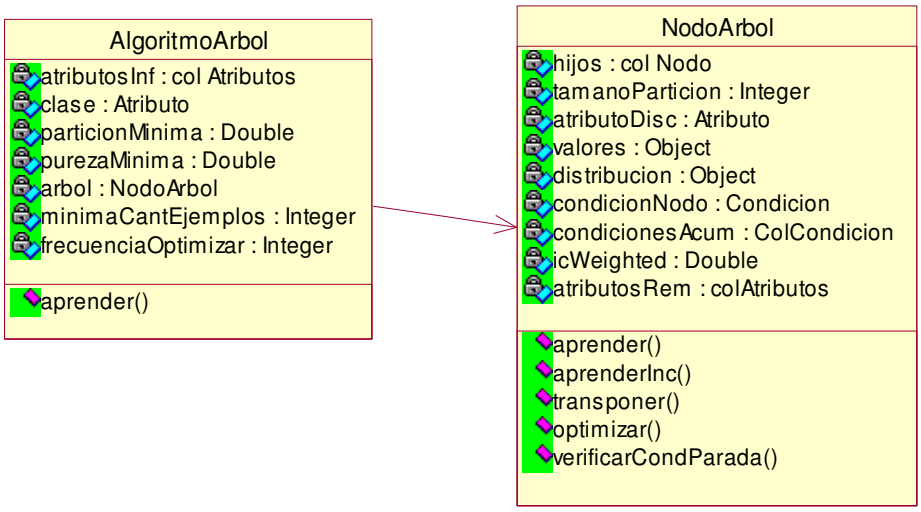


Figura 6.3 Subsistema de aprendizaje

AlgoritmoArbol: este objeto tiene el objetivo de almacenar los parámetros del algoritmo e implementar los métodos que se refieran a la estructura árbol.

Variable de instancia

AlgoritmoArbol>atributosInf: almacena la colección de atributos de inferencia.

AlgoritmoArbol>clase: almacena el atributo clase.

AlgoritmoArbol>particionMinima: esta es una condición de parada del algoritmo. Cuando el tamaño de la partición es menor a este valor, el algoritmo no subdivide más por considerarlo de poco valor estadístico.

AlgoritmoArbol>purezaMinima: esta es una condición de parada del algoritmo. Cuando el porcentaje de ejemplos pertenecientes a una clase supera este valor, el algoritmo no subdivide más por considerar pura la partición.

AlgoritmoArbol>nodoArbol: esta variable almacena un puntero al nodo raíz del árbol.

AlgoritmoArbol>minimaCantEjemplos: representa la cantidad mínima de ejemplos necesarios para crear la estructura.

AlgoritmoArbol>frecuenciaOptimizar: representa el intervalo de ejemplos necesarios para efectuar una optimización del árbol.

Métodos

AlgoritmoArbol>aprender: unCaso
Objetivo: incorporar el efecto del ejemplo de parámetro en la estructura árbol.
Pseudocódigo

Árbol está creado?
Si: nodo (raíz) aprenderInc: unCaso.
Cumple el intervalo de optimización?
Si: nodo (raíz) optimizar.
Agregar el ejemplo en el repositorio de casos.
No: Cumple la cantidad mínima de ejemplos?
Si: Agregar el ejemplo en el repositorio de casos.
Nodo (raíz) aprender.

NodoArbol: este objeto representa los nodos internos y hojas del árbol. La mayor carga algorítmica se aplica en este objeto. Por este motivo se incluye la descripción de los principales métodos implementados.

Variables de instancia

NodoArbol>hijos: almacena el puntero a los nodos hijos. Cada nodo hijo representa un valor lingüístico. Cada rama construida significa una restricción fuzzy.

NodoArbol>tamanoParticion: almacena la cantidad de ejemplos de aprendizaje contenidos en el nodo. Este valor puede ser un número real.

NodoArbol>atributoDisc: almacena el atributo elegido para realizar la partición del nodo. Los nodos hojas tienen nulo en este parámetro. Es esencial para recorrer la estructura.

NodoArbol>atributosRem: almacena la lista de atributos que todavía no se utilizaron como atributos discriminadores. El algoritmo permite que cada atributo sea "atributo discriminador" sólo una vez en cada camino desde la raíz hasta las hojas.

NodoArbol>valores: esta variable almacena por cada atributo remanente una matriz. Dicha matriz contiene en las filas los valores lingüísticos del atributo remanente y en la columna los valores lingüísticos del atributo clase. En cada celda se guarda la cantidad de ejemplos que cumplen las condiciones acumuladas y la condición de la celda (fila, columna). La implementación de esta variable se efectúa con un diccionario.

NodoArbol>distribución: almacena la proporción de ejemplos pertenecientes a cada clase. Esta variable se utiliza para determinar una condición de parada.

NodoArbol>condicionNodo: representa la restricción fuzzy añadida por la rama que llega hasta el nodo. Dicha restricción tiene la forma {variable lingüística = valor lingüístico}.

NodoArbol>condicionesAcum: representa las restricciones acumuladas desde el nodo raíz hasta la hoja. Esta variable acelera el pasaje a reglas de la estructura árbol.

NodoArbol>icWeigthed: almacena la información promedio contenida en el nodo. La información promedio es la métrica utilizada por el algoritmo para definir el atributo discriminador.

Métodos

NodoArbol>aprender
Objetivo: armar la estructura árbol desde cero.
Pseudocódigo

```
Nodo cumple condiciones de parada? (No se debe expandir más el árbol).
  Si: Retorna nodo
attrGanador:= nodo determinaAtributoGanador.
Nodo atributoDisc:= attrGanador.
Por cada fuzzy set fz de attrGanador hacer:
        CondNodo:= crear condición con atributo: attrGanador fuzzySet: fz.
        nodoHijo:= crear nodo condicionNodo: CondNodo.
        nodoHijo aprender (llamada recursiva).
Fin cada.
```

NodoArbol>aprenderInc: unCaso
Objetivo: incorporar el efecto del nuevo ejemplo de aprendizaje en la estructura. La estructura existente se reutiliza.

Pseudocódigo

```
gradoCor:= Calcular grado correspondencia de la condición del nodo con el caso de parámetro.
gradoCor > 0? (evita cálculos innecesarios).
  Si:
        Nodo recalcularTablasNodo: unCaso
        Nodo llenarDistribucion.
        Nodo setearTamParticion.
  No: retorna.
Nodo es hoja?
  Si:   Nodo cumple condiciones de parada?
              Si: Retorna.
              No: Nodo expandirNodo: unCaso.
Por cada hijo n de nodo hacer:
  N aprenderInc: unCaso (algoritmo recursivo)
Fin cada.
```

NodoArbol>optmizar
Objetivo: asegurar que el atributo de discriminación sea el adecuado para la heurística de selección, y que las condiciones de parada se cumplan sólo en las hojas.
Pseudocódigo

```
Nodo cumple condiciones de parada?
  Si:
        Podar hijos del nodo (transformarlo en hoja).
AtrGanador:= Calcular atributo ganador.
Nodo AtributoDisc <> atrGanador?
  Si:   Nodo transponer: atrGanador.
        Por cada hijo n de nodo hacer:
              N optimizar (algoritmo recursivo)
```

Fin cada.
No:
Por cada hijo n de nodo hacer:
N optimizar (algoritmo recursivo)
Fin cada.

NodoArbol>transponer: unAtributo
Objetivo: instalar el atributo de parámetro como atributo discriminador del nodo. El proceso se realiza cuidando reutilizar los conteos almacenados en los nodos. Las condiciones acumuladas para llegar a cada nodo desde la raíz se mantienen. Las variables que almacenan los conteos deben recalcularse.

NodoArbol>verificarCondParada
Objetivo: verificar si las condiciones de parada fijadas como parámetro se cumplen en el nodo.
Pseudocódigo

Nodo tamanoParticion < algoritmoArbol particionMinima?
Si: Retornar verdadero.
Nodo tamanoParticion < algoritmoArbol particionMinima?
Si: Retornar verdadero.
Nodo lista de atributos remanentes está vacía?
Si: Retornar verdadero.
Nodo pureza > algoritmoArbol purezaMinima? (la pureza del nodo se calcula con la variable de instancia nodoArbol>distribución)
Si: Retorna verdadero.
Retorna falso.

6.4.1 Funcionamiento

El objeto AlgoritmoArbol almacena los parámetros del algoritmo. Cuando existe suficiente evidencia para construir el árbol de decisión, el algoritmoArbol llama al método aprender del nodo raíz. La evidencia nueva se incorpora convocando al método "aprenderInc" del nodo raíz. Cuando se cumple el intervalo de optimización, el AlgoritmoArbol llama al método optimizar del nodo raíz.

6.5 Inferencia

El subsistema de inferencia consta de dos objetos: la base de reglas y las reglas propiamente dichas. El objetivo del mecanismo de inferencia es obtener, en forma no determinística, el valor del aspecto de contexto de alto nivel que le corresponde a los aspectos de contexto atómicos tomados desde los sensores.

La base de reglas está formada por reglas que tienen un antecedente y un consecuente, este último representado por el atributo clase. El antecedente de cada regla está constituido por condiciones unidas por el operador de conjunción (AND). Una condición está formada por el par [variable lingüística - valor lingüístico].

El subsistema consta de las siguientes clases (*figura 6.4*):

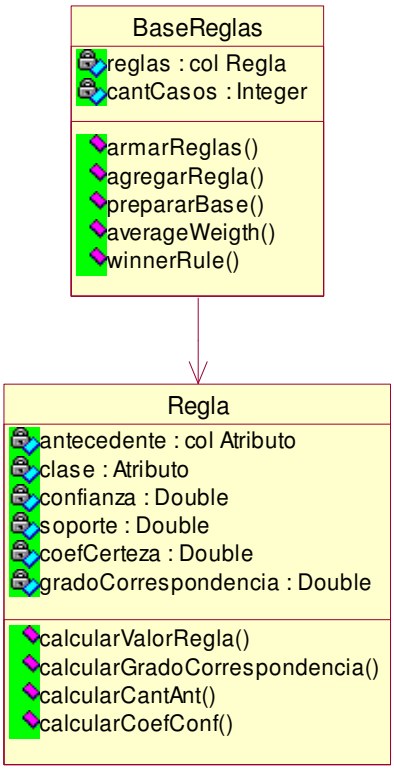


Figura 6.4 Subsistema de inferencia

BaseReglas: esta clase tiene como objetivo almacenar las reglas que conforman el modelo de aprendizaje e inferir el valor del atributo clase.

Variables de instancia

BaseReglas>reglas: esta variable almacena las reglas contenidas en la base.
BaseReglas>cantCasos: esta variable almacena la cantidad de ejemplos de aprendizaje.

Métodos

BaseReglas>armarReglas: unArbol
Objetivo: convertir la estructura árbol en un conjunto de reglas.
Pseudocódigo

PrepararBase: unArbol.
Por cada hoja de unArbol hacer:
  soporte:= calcular soporte de la regla.
  confianza:= calcular confianza de la regla.
  condicionesAcum:= hoja condicionesAcum.
  factorCerteza := calcular factorCerteza.
  Crear regla con soporte, confianza, factorCerteza, clase y condicionesAcum.
Fin cada hoja.

BaseReglas>agregarRegla: unaRegla **clase**: unaClase.
Objetivo: agregar una regla dentro de la colección de reglas pertenecientes a una clase.

BaseReglas>prepararBase: unArbol
Objetivo: inicializar las variables necesarias de la base de reglas.

BaseReglas>averageWeigth: unCaso
Objetivo: calcular la clase correspondiente a un caso dado. Este valor se infiere mediante las reglas presentes en la base.
Pseudocódigo

Por cada regla perteneciente a una clase hacer:
gradoCorrespondencia:= calcular grado de correspondencia de la regla con unCaso.
Acumular gradoCorrespondencia.
Fin cada regla.
Retorna el par [valor lingüístico clase – grado de correspondencia acumulado] por cada valor lingüístico del atributo clase.

Regla: esta clase tiene como objetivo almacenar los parámetros que constituyen cada regla, y calcular el grado de correspondencia con un ejemplo de clasificación.

Variables de instancia

Regla>antecedente: esta variable contiene las restricciones fuzzy que constituyen el antecedente de la regla.

Regla>clase: esta variable contiene el atributo clase, consecuente de la regla.

Regla>soporte: esta variable contiene el soporte calculado de la regla.

Regla>confianza: esta variable contiene la confianza de la regla.

Regla>coefCerteza: esta variable contiene el factor de certeza de la regla.

Regla>gradoCorrespondencia: esta variable contiene el grado de correspondencia de la regla con el ejemplo a clasificar.

Métodos

Regla>calcularGradoCorrespondencia: unCaso
Objetivo: calcular el grado de correspondencia entre el caso de parámetro y la regla.
Pseudocódigo

correspAnt:= unCaso calcularConteoInd: (antecedente) **tnorm**: (Producto new).
factorCerteza:= regla coefCerteza.

Retornar (correspAnt*factorCerteza) (suponiendo que el operador de conjunción utilizado sea el t-norm Producto).

6.5.1 Funcionamiento

La base de reglas recibe la estructura árbol resultado del aprendizaje y procede a armar el sistema de reglas correspondiente. Las reglas se ordenan por clase para mejorar el proceso de inferencia. Cada vez que se crea una regla se definen sus métricas más importantes (soporte, confianza y factor de certeza). Dichas métricas se obtienen de las hojas del árbol.

6.6 Repositorio de casos

El repositorio de casos es un objeto auxiliar que se encarga de mantener los casos completos y brindar servicios de conteo al algoritmo de aprendizaje.

Un caso es completo cuando tiene presente el valor para cada atributo de inferencia y el valor del atributo clase. Los casos para inferencia, armados por el control de sensores, carecen del valor del atributo clase, ya que el mismo se establece en la fase de inferencia.

El subsistema consta de los objetos: repositorio de casos y caso (*figura 6.5*).

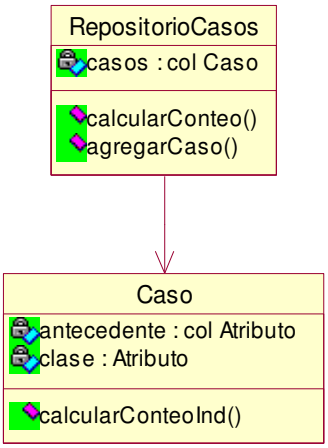


Figura 6.5 Subsistema repositorio de casos

RepositorioCasos: esta clase almacena los ejemplos de aprendizaje y provee de servicios de conteo global (referidos a todos los casos de aprendizaje).

Variables de instancia

RepositorioCasos>casos: esta variable almacena el puntero a los ejemplos de aprendizaje.

Métodos

RepositorioCasos>agregarCasos: unCaso
Objetivo: agregar un ejemplo de aprendizaje al repositorio.

RepositorioCasos>calcularConteo: unaColeccionCondiciones **tnorm**: unTnorm
Objetivo: calcular la cantidad de casos que cumplen las condiciones especificadas por el parámetro unaColeccionCondiciones. El operador de conjunción utilizado será el especificado en el parámetro unTnorm.
Pseudocódigo

Por cada caso hacer:
cantCasos:= caso clacularConteoInd: unaColeccionCondiciones tnorm: unTnorm.
Acumular cantCasos.
Fin cada caso.
Retornar cantidad de casos acumulado.

Caso: este objeto almacena los atributos que componen el ejemplo de aprendizaje. El mismo está formado por una colección de atributos de inferencia y un atributo clase.

Variables de instancia

Caso>antecedente: esta variable almacena la colección de atributos que conforman el antecedente de la regla.

Caso>clase: esta variable almacena el atributo clase.

Métodos

Caso>calcularConteoInd: unaColeccindCondiciones **tnorm**: unTnorm.
Objetivo: calcular la parte del ejemplo que cumple con las restricciones suministradas. El valor estará en el rango [0,1].
Pseudocódigo

TotalFuzzySets:= lista inicializar.
Por cada condicion COND en unaColCondiciones hacer:
AttrCond:= Cond atributo.
FuzzySet:= Cond fuzzySet.
FuzzySetCaso:= buscar fuzzy set en caso, con atributo AttrCond y fuzzy set FuzzySet
Agregar FuzzySetCaso en lista TotalFuzzySets.
Fin cada.
TotalConteo:= TotalFuzzySets tnorm: unTnorm.
Retornar TotalConteo.

6.6.1 Funcionamiento

El algoritmo de aprendizaje envía al repositorio de casos una colección de condiciones cada vez que necesita evaluar una partición, el método calcular conteo del repositorio de casos implementa esta funcionalidad. Además de la colección de

condiciones, el método recibe como parámetro un "Tnorm". Como se explicó en los capítulos preliminares, un Tnorm es un operador de conjunción utilizado en lógica difusa. El sistema soporta los dos Tnorms más utilizados en lógica difusa, ellos son el operador producto y el operador mínimo.

El repositorio de casos envía a cada caso el mensaje calcular conteo individual, teniendo como parámetro la colección de condiciones y un tnorm. El caso implementa esta funcionalidad calculando el grado de correspondencia entre el caso y la colección de condiciones enviada. La suma de los grados de correspondencia de cada caso con la colección de condiciones enviadas, será la respuesta del método calcular conteo del repositorio de casos.

6.7 Atributos y fuzzySets

Los atributos se dividen en dos tipos: los atributos de inferencia y el atributo clase. Los atributos de inferencia son las variables que se usarán para inferir el valor del atributo clase.

Un caso estará constituido por una instancia de cada atributo de inferencia y una instancia de un atributo clase.

El algoritmo adoptado en la clasificación utiliza lógica difusa, por esta razón cada instancia de un atributo de inferencia almacena el valor "crisp" que proviene de los sensores y su correspondiente valor fuzzy. El valor fuzzy está constituido por los fuzzy sets definidos para el atributo.

Cada fuzzy sets de un atributo tiene asociado el grado de membresía que le corresponde al valor "crisp" almacenado en el atributo. Cada atributo tiene un valor máximo y un valor mínimo que se corresponde con el dominio del mismo. Estos valores se usarán para la validación del valor procedente de los sensores.

El sistema soporta las dos formas más utilizadas para modelar los fuzzy sets, ellas son: forma trapezoidal y forma triangular. La diferencia entre ambas es que la forma trapezoidal permite un grado de membresía total (grado de membresía = 1) para un rango de valores del atributo.

La *figura 6.6* muestra las clases atributo y fuzzySets

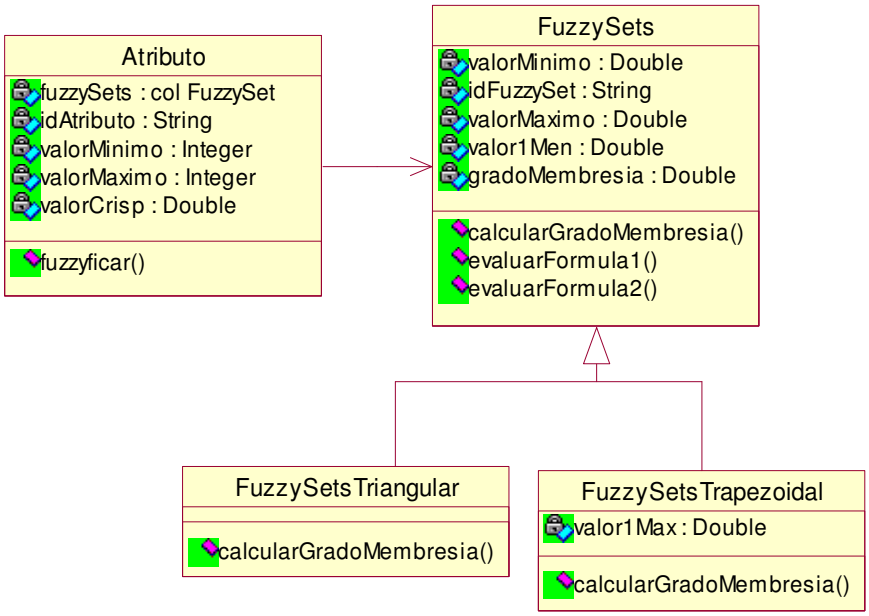


Figura 6.6 Atributos y fuzzy Sets

Atributo: este objeto representa las variables lingüísticas o aspectos de contexto que forman parte del sistema.

Variables de instancia

Atributo>fuzzySets: almacena la colección de fuzzy sets asociada. Cada fuzzy set se corresponde con un valor lingüístico.

Atributo>idAtributo: almacena el nombre del atributo.

Atributo>valorMinimo: almacena el valor mínimo del atributo.

Atributo>valorMáximo: almacena el valor máximo del atributo

Atributo>valorCrisp: almacena el valor tomado desde el sensor.

Métodos

Atributo>fuzzyficar: unValorCrisp
Objetivo: calcular el grado de membresía de cada fuzzy set asociado con el valor recibido en el parámetro.

FuzzySets: este objeto almacena la definición de la partición fuzzy hecha en el atributo.

Variables de instancia

FuzzySets>idFuzzySet: esta variable almacena la identificación del valor lingüístico.

FuzzySets>valorMinimo: almacena el límite mínimo.

FuzzySets>valorMaximo: almacena el límite máximo.

FuzzySet>valor1Men: establece el valor que tiene máximo grado de membresía (1). Los fuzzy sets triangulares tienen definido sólo este valor. Los fuzzy sets trapezoidales tienen además el valor máximo en el cual es máximo el grado de membresía.

FuzzySet>gradoMembresia: almacena el grado de membresía que le corresponde al valor sensado.

Métodos

FuzzySet>calcularGradoMembresia: unValorCrisp
Objetivo: calcular el grado de membresía correspondiente al valor de parámetro. Cada forma sobrescribe este método.

FuzzySet>evaluarFormula1: unValorCrisp
Objetivo: calcular el grado de membresía correspondiente al valor de parámetro, en el rango que la función asciende.

FuzzySet>evaluarFormula2: unValorCrisp
Objetivo: calcular el grado de membresía correspondiente al valor de parámetro, en el rango que la función desciende.

6.8 Control principal

El control principal es el objeto que coordina el funcionamiento del sistema, funciona como un "Mediator" en los términos de patrones de diseño de [Gamma 1995], [Alpert 1998] .

Para cumplir su propósito almacena en sus variables de instancia los objetos: "Algoritmo árbol", "Control de sensores", "Base de reglas", "Repositorio de casos", los "Atributos de inferencia" y "Atributo clase". Todos estos objetos, excepto los atributos de inferencia y el atributo clase, se crean una sola vez y forman la estructura estable del sistema.

La *figura 6.7* muestra la arquitectura del Control principal

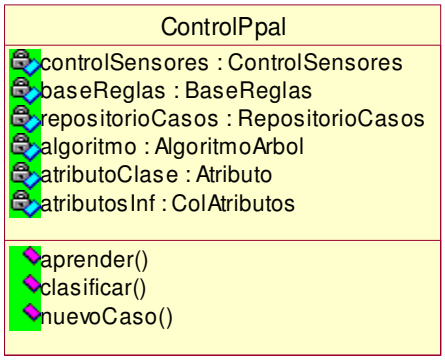


Figura 6.7 Control principal

Variables de instancia

ControlPpal>controlSensores: almacena el puntero al control de sensores.

ControlPpal>baseReglas: almacena el puntero a la base de reglas.

ControlPpal>repositorioCasos: almacena el puntero al repositorio de casos.

ControlPpal>algoritmo: almacena el puntero al algoritmo de aprendizaje, en este caso sólo se implementa un algoritmo basado en árbol de decisión.

ControlPpal>atributosInf: almacena la colección de atributos de inferencia.

ControlPpal>atributoClase: almacena el atributo clase.

Métodos

ControlPpal>nuevoCaso: unCaso
Objetivo: enviar el caso de parámetro a la base de reglas para su clasificación. Esta notificación la recibe desde el control de sensores.

ControlPpal>clasificar: unCaso **metodo**: unMetodo
Objetivo: enviar el caso de parámetro a la base de reglas utilizando el método de inferencia elegido.

ControlPpal>aprender: unCaso
Objetivo: llamar al algoritmo de aprendizaje para agregar el efecto del ejemplo completo (contiene la clase). Luego de agregar el efecto del ejemplo de aprendizaje, este se almacena en el repositorio de casos.

6.8.1 Funcionamiento

El control principal mantiene un mecanismo de dependencia con el control de sensores. Cuando el control de sensores envía un caso al control principal, este último

ejecuta el método “nuevo caso” teniendo como parámetro el caso. Este método llama al método clasificar con el caso y el método de inferencia como parámetros.

El método clasificar del control principal devuelve los valores lingüísticos del atributo clase, cada uno tiene asociado el grado de correspondencia.

Luego de recibir la clase correcta para cada ejemplo, el control principal llama al algoritmo de aprendizaje (en este caso el AlgoritmoArbol) con el nuevo ejemplo como parámetro.

Capítulo VII Integración con framework Context Aware

7.1 Introducción

En este capítulo el foco estará puesto en la integración del sistema de inferencia con el framework Ballon [Fortier 2005]. La idea subyacente es enriquecer la utilidad del framework incorporando aspectos de contexto derivados no determinísticos.

Para evitar confusiones se tomará la siguiente convención: framework se refiere al framework Ballon, sistema de inferencia se refiere al sistema desarrollado en esta tesis.

Los puntos fuertes del framework son:

- La separación en capas de software. Esto permite el cambio de alguna capa sin que las otras se vean involucradas. Las capas definidas son: la capa de aspectos de sensado, la capa de aspectos de contexto y la capa de servicios.
- El desarrollo de un mecanismo de dependencias basado en eventos, especializado para aplicaciones sensibles a contexto.
- La interpretación de un aspecto de contexto como un objeto con funcionalidad propia y claramente definida.
- La noción de separación del "contexto" en objetos atómicos llamados aspectos de contexto, junto con la visión de considerar como no acotado el contexto de una aplicación. Esto permite fácilmente añadir o eliminar aspectos de contexto relevantes, cada uno con su respectiva funcionalidad.

7.2 Vista preliminar framework

Aspectos de sensado

En el framework se distinguen varias capas de software que interactúan entre sí para obtener una estructura flexible. Las capas se comunican mediante un mecanismo de dependencia semejante al patrón Observer de [Gamma 1995]. El siguiente diagrama muestra la arquitectura general del framework para una aplicación location-aware (*figura 7.1*)

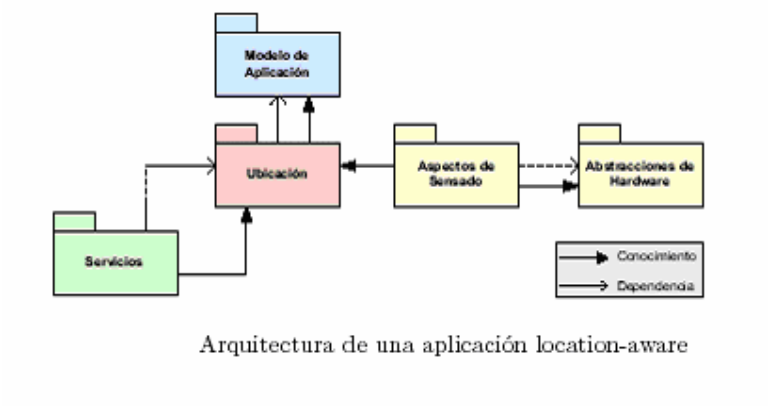


Figura 7.1 Arquitectura general framework

Modelo de aplicación

El modelo de aplicación tiene incorporado el comportamiento habitual de la aplicación. Este comportamiento de la aplicación estándar es el que se va a enriquecer con el comportamiento sensible al contexto. La separación entre el modelo de la aplicación y el modelo sensible a contexto se hace para no tener que cambiar los objetos residentes en la aplicación "fuente". Algunos de los objetos de la aplicación tendrán el comportamiento definido originalmente y se le agregará el comportamiento sensible al contexto.

La clase de la aplicación a la cual se le agrega comportamiento sensible a contexto se denomina en forma general "usuario". El usuario puede ser una persona, un automóvil, etc. La aplicación sensible a contexto conoce y es dependiente de dicho objeto.

Capa de ubicación

El framework se creó inicialmente para generar aplicaciones sensibles a la ubicación. Luego, los conceptos establecidos para el aspecto de contexto "locación" se extendió para otros aspectos de contexto. Por lo tanto podríamos generalizar el nombre de esta capa como capa "aspectos de contexto". En esa capa se agregan los objetos que representan los aspectos de contexto. Por ejemplo, en una aplicación sensible a la ubicación de una universidad, el aspecto locación estaría representado por un aula. De esta manera cuando el usuario, en este ejemplo podría ser un alumno, ingresa a un aula determinada se le brinda una serie de servicios predeterminados.

El objeto que representa un aspecto de contexto puede tener su contraparte en el modelo de la aplicación o no. Podría ocurrir que el objeto aspecto de contexto sea irrelevante para el modelo de aplicación, pero que surja la necesidad de hacer reaccionar al sistema dependiendo del estado de dicho objeto. Por ejemplo, en el caso del sistema de la universidad, los pasillos no tienen relevancia en la aplicación fuente, pero es importante para el comportamiento sensible a contexto. En caso que el objeto aspecto de contexto tenga vinculación o represente a un objeto del modelo de la aplicación, el objeto aspecto de contexto debe conocer y ser dependiente de su contraparte.

Servicios

En esta capa se encuentra codificada la funcionalidad sensible al contexto. Este comportamiento es el que se va a añadir a los objetos de la capa "modelo de la aplicación". Como indica la figura, los servicios dependen de los aspectos de contexto que configuran globalmente el "contexto" de la aplicación. Por este motivo, los servicios conocen y son dependientes de los aspectos de contexto que se tienen en cuenta. En forma general, la capa de servicios estará conformada por las siguientes clases:

Clase servicio: en esta clase se representarán los servicios disponibles para el usuario. Es interesante notar que el comportamiento sensible a contexto se modela como un objeto y se denomina servicio. Un ejemplo de servicio en el sistema de la universidad sería presentar al alumno que ingresa a un aula (en su dispositivo móvil) la bibliografía de consulta señalada por el docente.

Clase controladora: esta clase tiene la responsabilidad de administrar y coordinar el manejo de la capa. Por ejemplo, cuando cambia un aspecto de contexto de un usuario, la clase controladora debe reasignar los servicios brindados al mismo.

Clase proveedora de servicios: esta clase puede verse como un "contenedor" de servicios. Esta trabaja como objeto intermediario entre los servicios disponibles y los aspectos de contexto. Se podría ver como una serie de restricciones que se deben cumplir para que un usuario obtenga los servicios programados. Por ejemplo, en una aplicación sensible solamente a la locación, la clase estaría formada por todas las posiciones relevantes para otorgar servicios al usuario.

Clase "usuario": esta clase representa al usuario desde el punto de vista de los servicios. Es el objeto que recibe el comportamiento sensible al contexto. Este representa al usuario del modelo de la aplicación.

7.3 Descripción framework

7.3.1 Aspectos de sensado

Esta capa tiene la misión de interpretar el valor "crudo" que proviene de los sensores. Los sensores están incluidos en la capa de abstracciones de hardware. Una vez que se ha capturado la información externa a la aplicación, ésta debe ser convertida en objetos representantes de los aspectos de contexto.

Cada aspecto de contexto conoce y es dependiente de los cambios de estado producidos en los sensores.

El funcionamiento de esta capa se resume en tres pasos, ellos son:

- Capturar la información disparada por el sensor conectado al aspecto de contexto. Esta funcionalidad se consigue mediante un mecanismo de dependencia.
- Convertir el valor "crudo", obtenido en el paso anterior, en un objeto que pertenezca al dominio del aspecto de contexto asociado. Por ejemplo, si el aspecto de contexto es la temperatura y sus valores posibles son: Alta, Media y Baja; el valor recibido de un termómetro (el sensor en este caso) debe ser convertido en los objetos Alta, Media o Baja. Es necesario que los valores del aspecto de contexto asociado sean objetos para contener comportamiento. El comportamiento que se debe incluir en la mayoría de los casos es la "Igualdad".
- Notificar al aspecto de contexto asociado el cambio de estado producido.

La clase que administra toda la funcionalidad de la capa es el SensingConcern. Esta última interactúa con dos clases de apoyo, estas son:

SensingPolicy: esta clase tiene la funcionalidad de comunicarse con el sensor de una manera determinada. En el framework se proveen dos políticas de comunicación con los sensores:

- **Pull**: cuando se utiliza esta política, la clase debe consultar con una cierta periodicidad el estado del sensor.

- **Push**: cuando se utiliza esta política, la clase toma un rol pasivo, espera que el sensor le comunique un cambio de estado para desencadenar el proceso citado anteriormente.

Dispatcher: esta clase se encarga de validar el envío del valor recogido por el sensor. El framework propone dos tipos de dispatcher: el dispatcher "Forward" y el dispatcher "ErrorFilter". El "forward" simplemente reenvía el valor recibido por el sensor sin procesamiento alguno; en cambio, el "ErrorFilter" valida el dato recibido.

La *figura 7.2* muestra el diagrama de clases que componen la capa. En este caso se muestra a la locación como el único aspecto de contexto que se tiene en cuenta en la aplicación.

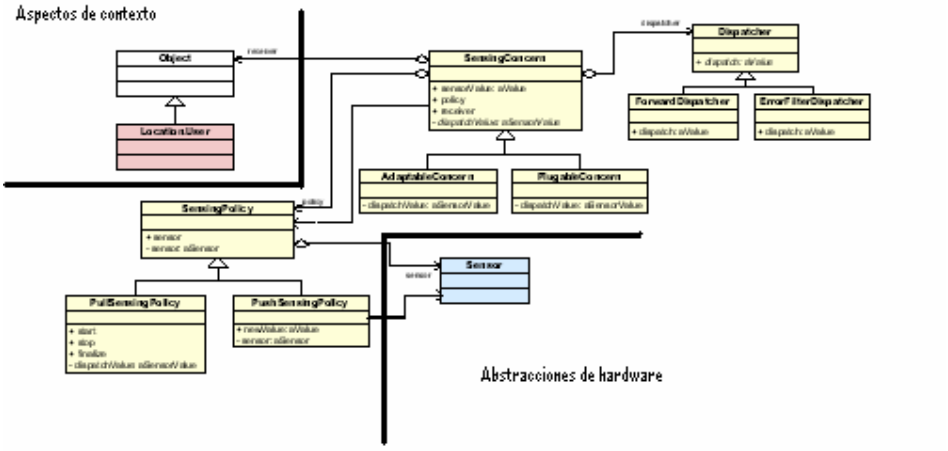


Figura 7.2 Capas aspectos de contexto, aspectos de sensado y abstracciones de hardware.

7.3.2 Abstracciones de hardware

En esta capa se encuentran las clases que representan los dispositivos de hardware utilizados para la adquisición de datos. La capa tiene muy bajo nivel de abstracción, ya que se trabaja con los controladores provistos por los fabricantes. El comportamiento incluido en estas clases se limita a distribuir y notificar los cambios producidos.

Los mecanismos de adquisición de datos no tienen conocimiento sobre la manera que se utilizará el mismo. La interpretación de los valores captados se efectúa en la capa de aspectos de sensado. Esta separación de comportamiento permite reutilizar los sensores en otra aplicación con distinta interpretación de datos.

A manera de ejemplo se menciona algunos de los sensores que se pueden utilizar en esta capa: sensor infrarrojo, receptor de GPS, termómetro, cronómetro, etc.

7.3.3 Capa de servicios

La capa de servicios es la parte más compleja del framework. Esto es así porque, como se dijo anteriormente, en la capa de servicios se incluye la funcionalidad sensible

al contexto. Este comportamiento es el que se agrega a la funcionalidad provista por la aplicación original.

En el framework los servicios se modelan como objetos que deben implementar un conjunto de responsabilidades básicas.

La clase que representa a los servicios, UserService, tiene dos tipos de mensajes: mensajes concretos y mensajes abstractos. Los mensajes abstractos son "Hooks" que provee el sistema y se debe implementar en las clases concretas de servicio. Los mensajes concretos se utilizan para el manejo interno del framework. Por ejemplo, la orden de activar o desactivar un servicio es un manejo interno del framework, en cambio, la manera que un servicio se convierte en activo depende de cada servicio en particular.

Esta capa administra los servicios disponibles y activos para un usuario. Para lograr este propósito cuenta con una representación del usuario desde el punto de vista de los servicios, la clase Service.User. Esta clase está relacionada con los aspectos de contexto de los cuales es dependiente. Esta es la conexión entre la capa de servicios y la capa de aspectos de contexto.

La interacción entre los usuarios y los servicios asociados sucede en un ambiente de servicios. Este entorno se representa por medio de la clase ServiceEnvironment, siendo su responsabilidad llevar cuenta de los servicios disponibles, de los usuarios que se encuentran activos y de mediar entre los distintos componentes del framework al producirse un cambio en el contexto del usuario.

Cuando implementamos una aplicación sensible a contexto debemos definir el comportamiento que tendrá, cuando se cumplan una serie de restricciones expresadas en términos de los aspectos de contexto considerados. Por ejemplo, en un modelo sensible a la ubicación, necesitamos especificar los servicios que se le debe brindar a un usuario cuando su ubicación (aspecto de contexto locación) sea un aula determinada. La clase ServiceProvider cumple el objetivo de evaluar las restricciones impuestas para que se le asignen ciertos servicios a un usuario. La clase contiene un conjunto de restricciones, representadas por la clase abstracta Constraints. Cada aspecto de contexto debe generar sus propias clases concretas para indicar en qué caso los servicios pueden utilizarse por el usuario.

El grafico siguiente muestra la relación entre los servicios, los aspectos de contexto y el modelo de aplicación (*figura 7.3*).

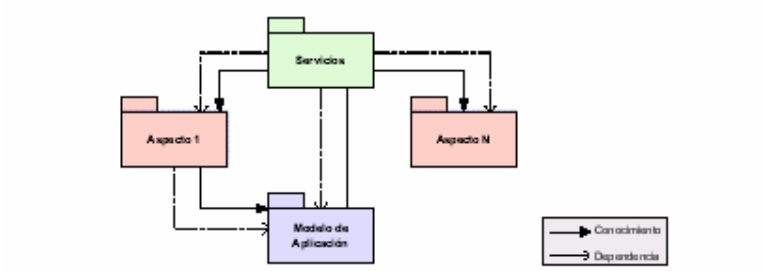


Figura 7.3 Relación entre la capa de servicios, el modelo de la aplicación y los aspectos de contexto.

7.3.3.1 Manejo de eventos especializado

El ambiente en el que se encuentra codificado el framework (Smalltalk), posee un mecanismo de dependencia basado en símbolos. Por ejemplo, un aspecto de contexto

avisa que cambió su estado enviando el nombre del mensaje como un símbolo; en caso del aspecto de contexto locación el mensaje será self changed: #location.

Si el framework usara el método de dependencia mencionado anteriormente, requeriría que se modifique el ServiceEnvironment cada vez que se agregue o quite un aspecto de contexto a tener en cuenta. Esto significaría ingresar código del estilo "If aspecto = 'nombre de aspecto de contexto' Then acción X"; lo cual es difícil de mantener. Para evitar este inconveniente, el framework dispone de un mecanismo de dependencia especializado para aplicaciones sensibles a contexto.

El mecanismo de dependencia creado toma los cambios en el estado de cada aspecto de contexto como un objeto, de esta manera se pueden generalizar los eventos de cambio en una clase abstracta de la cual deriven las clases concretas para cada aspecto de contexto. Estas últimas tendrán encapsulado el comportamiento adecuado para tratar el cambio de estado en un aspecto de contexto.

El sistema de dependencias refinado comprende dos tipos de mensajes:

- **Eventos**: se utilizan para disparar eventos de cambio.
- **Manejadores**: se utilizan para indicar el comportamiento correspondiente a cada evento ocurrido.

Al ocurrir un cambio de estado en un aspecto de contexto, el framework creará un objeto "ContextEvent"; lo configurará con el nombre del evento, el objeto que disparó el cambio y los parámetros brindados por el emisor. Luego el mecanismo de dependencia avisa a los objetos interesados en el cambio.

Para cada aspecto de contexto que se agregue, habrá que agregar una dupla evento-manejador. De esta forma el ServiceEnvironment trata a todos los aspectos de contexto por igual, haciendo más robusto su diseño.

7.3.3.2 Funcionamiento de la capa de servicio

En este apartado se describe las clases componentes de la capa de servicio con un grado mayor de detalle.

La capa de servicios está compuesta por las siguientes clases:

ServiceEnvironment: esta clase cumple la labor de "manager" en la capa. Tiene las siguientes responsabilidades:

- Administrar los usuarios incluidos en la aplicación.
- Administrar los servicios disponibles.
- Administrar los proveedores de servicios.

UserService: es una clase que representa los servicios disponibles en el ambiente, funciona como un "modelo" para los servicios concretos. Por ejemplo, cada servicio debe conocer su nombre e implementar el comportamiento de activación, arranque y parada. La manera de implementar la activación depende de la naturaleza del servicio, si es una alarma, la activación significa indicarle un horario para que suene.

ServiceProvider: esta clase puede verse como un "contenedor" de servicios asociado a un conjunto de restricciones. Las restricciones deben estar hechas en términos de valores de aspectos de contexto. Por ejemplo, si se quiere asociar un proveedor de servicios con el aspecto de contexto locación, las restricciones deben formularse en términos del tipo locación. En el ejemplo de la universidad, un valor del

tipo locación sería un aula. Este diseño equivale a efectuar un "If" en un sistema procedural.

Constraint: esta jerarquía de clases se utiliza para formular las restricciones a las que está sujeto cada proveedor de servicios. Cada clase Restricción concreta debe implementar el mensaje "isSatisfiedBy: aUser"; la respuesta debe ser booleana. El framework provee la posibilidad de armar restricciones compuestas por varios aspectos de contexto. Este tipo de restricciones pueden estar vinculadas con el operador lógico "Or" o "And".

Service.User: esta clase representa al usuario desde el punto de vista de los servicios. La misma concentra el comportamiento sensible a contexto agregado al objeto que representa a un usuario. Tiene las siguientes responsabilidades:

- Administrar los controladores de eventos. Cada aspecto de contexto tiene asociado un evento de cambio, y cada evento de cambio tiene asociado un controlador.
- Administrar la suscripción a servicios. Esto se hace porque, en general, los usuarios no están interesados en todos los servicios disponibles, sólo les interesa un subconjunto de ellos.
- Administrar los aspectos de contexto a los cuales está sujeto el usuario. El objeto usuario es el que recibe la notificación de cambio en un aspecto de contexto activo por medio del mecanismo de dependencia.

EventHandler: esta clase abstracta funciona como "modelo" para los manejadores concretos de cada evento.

En el momento que se produce un cambio en algún aspecto de contexto, se desencadenan los siguientes procesos:

- El mecanismo de dependencia especializado crea un objeto "ContextEvent" que contiene el aspecto de contexto afectado, el objeto que disparó el cambio y opcionalmente parámetros relevantes al cambio.
- El Service.User recibe la notificación, busca el manejador del evento y avisa al ServiceEnvironment.
- El ServiceEnvironment activa el manejador indicándole el evento, el aspecto que lo generó y el ambiente de servicio en el que se está ejecutando.

El siguiente es el diagrama de clases *(figura 7.4)* que componen la capa de servicios y su relación con los aspectos de contexto (sólo se incluye el aspecto locación)

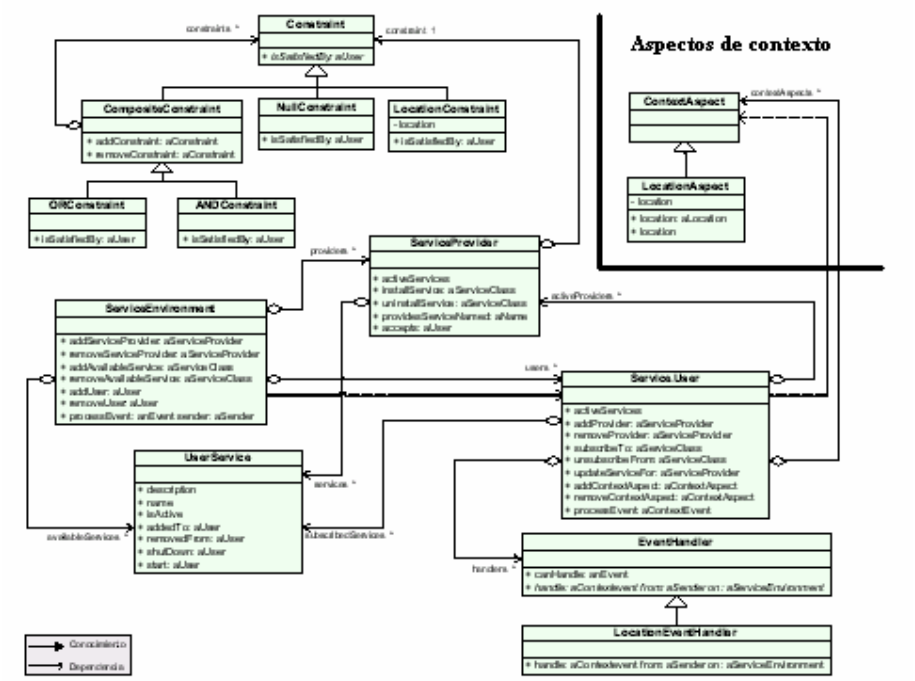


Figura 7.4 Capa de servicios

7.4 Conexión del sistema de inferencia con las clases del framework

En esta sección se mostrará la forma de conectar el sistema de inferencia (agente inteligente) descrito en los capítulos anteriores.

El aspecto de contexto no determinístico tiene un resultado nominal, por lo tanto puede tratarse de igual manera que un aspecto de contexto determinístico en la capa de servicios.

El sistema de inferencia puede verse como una capa de software intermedia entre la capa "Aspectos de sensado" y la capa "Aspectos de contexto". Los aspectos de contexto atómicos utilizados para inferir el aspecto de contexto de alto nivel, no están vinculados con el contexto del usuario; en cambio el aspecto de contexto inferido si lo está.

En la arquitectura del sistema de inferencia se incorporó la clase "Control de sensores" cuyas responsabilidades son:

- Administrar la relación con los sensores.
- Administrar el mecanismo de dependencias por cambio en los sensores.
- Administrar la relación entre los sensores y los atributos de inferencia (aspectos de contexto atómicos).

El modelo planteado es demasiado "simplista", ya que no considera problemas de bajo nivel tales como las políticas de sensado y el tratamiento de errores. Por este motivo es necesario cambiar las responsabilidades del "Control de sensores" y aprovechar las capas "abstracciones del hardware" y "aspectos de sensado" provistas por el framework.

7.4.1 Capa de software intermedia

El SensingConcern de la capa "Aspecto de contexto" convierte el valor recogido por el sensor en un objeto del dominio del aspecto de contexto relacionado. La lógica de conversión se efectúa mediante una estructura tipo "Diccionario", es decir, para cada valor sensado se obtiene unívocamente el valor correspondiente del aspecto de contexto tratado.

Para que la conexión entre el sistema de inferencia y las clases del framework sea efectiva, es evidente que se debe convertir al mecanismo de inferencia en "algo" cuya funcionalidad sea parecida a la de un SensingConcern. De esta manera resulta transparente para el framework el procedimiento por el cual se obtiene un aspecto de contexto a partir del valor sensado. Para generalizar la funcionalidad del SensingConcern se debe tener en cuenta que deberá operar con dos tipos de aspectos de contexto:

- **Aspecto de contexto determinístico**: convierte un valor sensado en un objeto aspecto de contexto por medio de una estructura tipo diccionario (la conversión es unívoca).
- **Aspecto de contexto no determinístico**: convierte valores sensados en un objeto aspecto de contexto mediante un algoritmo de inferencia (la conversión tiene incertidumbre).

De la explicación anterior surge la diferencia entre la semántica y funcionalidad entre un atributo y un aspecto de contexto. Un atributo tiene la funcionalidad que le impone el algoritmo de inferencia. Un aspecto de contexto debe tener la funcionalidad relacionada con la sensibilidad al contexto impuesta por el framework.

Para hacer portable el sistema de inferencia y permitir una evolución independiente del mismo, resulta clara la necesidad de no "contaminar" la funcionalidad de los objetos atributo y aspecto de aspecto. Sin embargo, al mismo tiempo debemos asociarlos de algún modo para que colaboren entre sí.

Una manera eficiente de separar la funcionalidad de los objetos, pero a su vez hacer que colaboren entre sí, es mediante el patrón de diseño "Decorator" [Gamma 1995].

La *figura 7.5* muestra la relación entre aspectos de contexto y atributos.

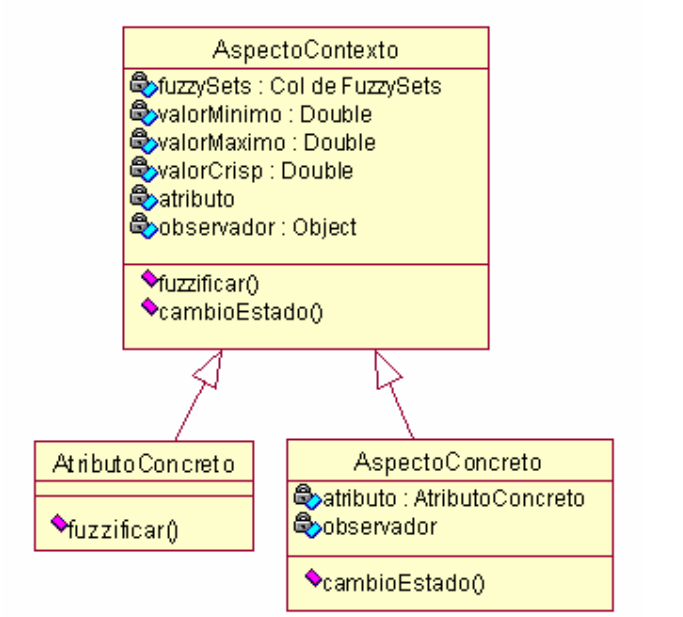


Figura 7.5 Relación aspectos de contexto y atributos

El rol de "decorador" lo cumple el aspecto de contexto concreto y el rol de decorado el atributo del sistema de inferencia. El aspecto de contexto tiene una variable de instancia que apunta al atributo relacionado. El aspecto de contexto delega toda funcionalidad que sea relativa a la inferencia, sólo implementa la funcionalidad sensible a contexto.

Existen dos tipos de atributos en el sistema de inferencia, ellos son:

- **Atributo de inferencia**: es necesario para efectuar la inferencia del atributo clase. No está relacionado con el "contexto" del usuario, es interno al sistema de inferencia.
- **Atributo clase**: es el resultado del sistema de inferencia. Está relacionado directamente con el contexto del usuario.

Con el esquema planteado, la conexión entre cada SensingConcern del framework y cada aspecto de contexto atómico quedaría de la siguiente forma (*figura 7.6*).

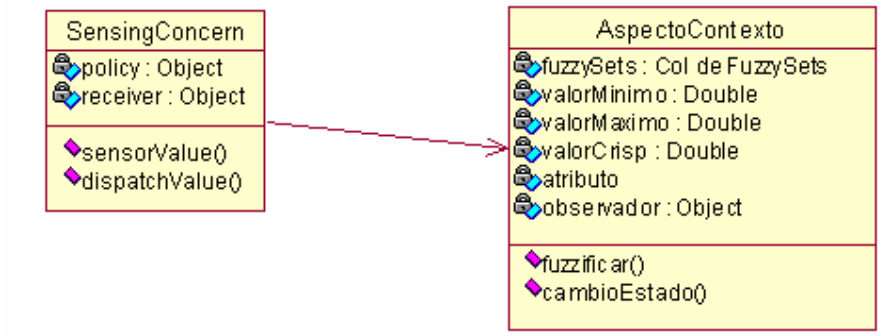


Figura 7.6 Relación entre aspecto de contexto y SensingConcern

7.4.2 Control de sensores

Dado que la funcionalidad de administrar los sensores pasa a formar parte de las clases del framework; el comportamiento del Control de sensores cambia ligeramente. Ahora este pasa a tener dos funcionalidades importantes:

- Administrar los cambios en los aspectos de contexto atómicos.
- Construir el "caso" para luego inferir la clase que le corresponde.

La relación del Control de sensores y los aspectos de contexto quedaría de la siguiente manera (*figura 7.7*):

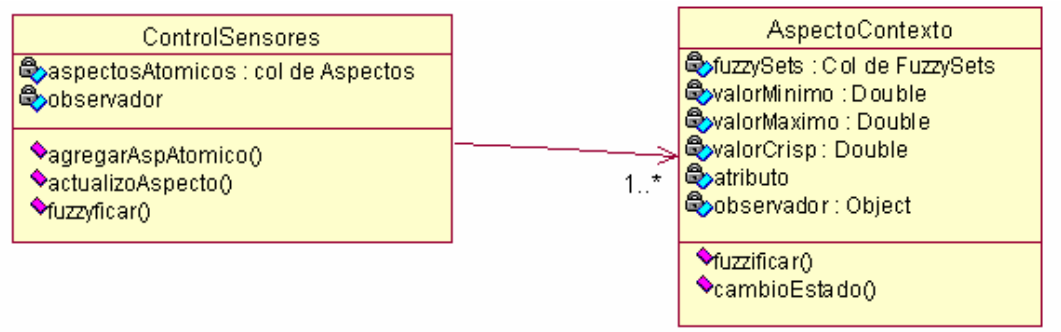


Figura 7.7 Relación Control de sensores y aspectos de contexto

7.4.3 Control principal

El Control de sensores del sistema de inferencia convierte la información proveniente de sensores en un "caso" a clasificar, y luego pasa el control al Control principal. Este último pasa el "caso" a la base de reglas y obtiene la clase que le corresponde. La clase es un valor del dominio del atributo clase y por ende del aspecto de contexto relacionado. A partir de ese momento el aspecto de contexto se comporta de manera semejante a cualquier aspecto de contexto determinístico.

La conexión entre el resultado del algoritmo de inferencia y las clases del framework queda de la siguiente manera (*figura 7.8*):

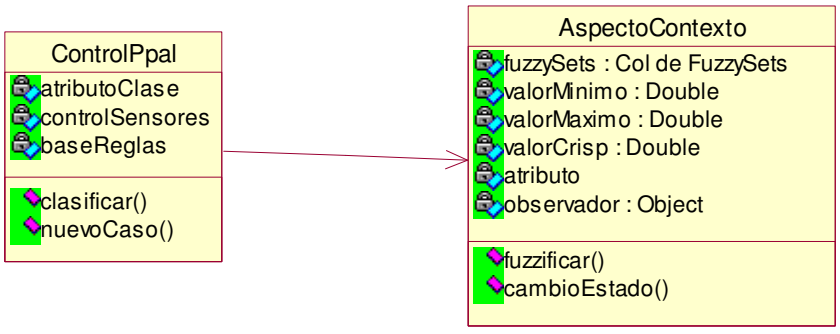


Figura 7.8 Relación Control Principal y aspectos de contexto

El atributo clase tiene la misma característica estructural que un aspecto de contexto de inferencia, es decir, las mismas variables de instancia.

7.4.4 Restricciones en la capa de servicios

La capa de servicios del framework dispone de una jerarquía de objetos Constraints, necesarios en la configuración de las restricciones que debe evaluar el ServiceProvider para otorgar servicios disponibles a los usuarios. El framework posibilita incorporar tres tipos de restricciones:

- **Simple**: se refiere a un aspecto en particular. Por ejemplo, en el aspecto de contexto Locación se puede ingresar la restricción Locación = "Aula 1".
- **Nula**: no es una restricción. Los servicios contenidos en el ServiceProvider se brindan al usuario en cualquier circunstancia. Por ejemplo, información general para toda persona que pase por la facultad.
- **Compuesta**: se refiere a la combinación de varios aspectos de contexto.

7.5 Definición de un problema ejemplo

En los apartados anteriores se hizo una introducción acerca del funcionamiento del framework especializado para construir aplicaciones sensibles a contexto.

La idea subyacente es aprovechar el framework para las tareas comunes en toda aplicación sensible a contexto y el sistema de inferencia para las tareas que requieran un mecanismo de inferencia. Una de la tareas no determinísticas requerida en este tipo de aplicaciones, es inferir el valor de un aspecto de contexto de alto nivel a partir de aspectos de contextos atómicos. Desde el punto de vista del desarrollador de una aplicación sensible al contexto, significa una extensión de alto valor agregado.

Para mostrar las posibilidades de integración entre el sistema de inferencia desarrollado y el framework especializado, se va a describir un problema.

Problema

Aspectos de contexto a considerar: Locación y Aptitud para actividades deportivas.

Locación: aspecto de contexto determinístico.

Aptitud para actividades deportivas: aspecto de contexto derivado no determinístico, inferido en base a los aspectos de contexto atómicos: Temperatura, Humedad y Presión atmosférica.

Servicio sujeto a restricciones: Turno de canchas.

Restricciones: aspecto de contexto Locación = "Patio de deportes" AND aspecto de contexto Aptitud para actividades deportivas = "Positivo".

7.5.1 Descripción del flujo de información

Antes de describir el flujo de información es necesario señalar las condiciones necesarias para la operación de la aplicación, es decir, las instancias más importantes de las clases del framework.

Condiciones iniciales

ServiceUser: representa al usuario en la capa de servicio. El usuario debe estar subscripto a los cambios producidos en los aspectos de contexto: Locacion y Aptitud para actividades deportivas.

UserService: representa los servicios disponibles en el ambiente. En este caso el servicio es "Turno de canchas".

ServiceEnvironment: es el objeto principal de la capa de servicios. El mismo conoce al ServiceUser, el ServiceProvider y el UserService.

ServiceProvider: es el contenedor de servicios sujeto a restricciones. En este caso contendrá al servicio "Turno de canchas", sujeto a la restricción compuesta por los aspectos de contexto: locación y aptitud para actividades deportivas.

EventHandler: es el manejador del evento que se produce cuando cambia el valor de un aspecto de contexto. Se supone que debe haberse creado una instancia para cada uno de los aspectos de contexto presentes en la aplicación: Locación y Aptitud para actividades deportivas.

Proceso

- **Sensor**: envía el valor leído a la capa Aspectos de sensado.
- **SensingConcern**: por medio del dispatcher reenvía el valor recibido al aspecto de contexto conectado al mismo.
- **Aspecto de contexto**: envía el valor a su "objeto decorado", el atributo de inferencia atómico relacionado, activando el mecanismo de dependencias del sistema de inferencia.
- **Control Principal**: por medio de este objeto, el sistema de inferencia devuelve el valor del aspecto de contexto derivado no determinístico (inferido).

El cambio de valor del aspecto de contexto inferido, desencadena el flujo normal de información en las clases del framework, tal como lo haría un aspecto de contexto determinista (por ejemplo, locación).

Capítulo VIII Estudio aplicación del sistema

8.1 Introducción

En este capítulo se va a describir y estudiar una de las posibles aplicaciones de la arquitectura y algoritmo descripto en esta tesis.

En la primera sección se describirán las condiciones generales en las cuales se desempeñará el software, relevando los aspectos más destacados a tener en cuenta cuando se aplica en un caso real. Luego se describirá un problema ejemplo que representa el objetivo de la tesis. Tomando como referencia el problema señalado, se harán análisis referentes al algoritmo y a la implementación.

8.2 Condiciones generales del problema

El modelo planteado en esta tesis tiene como objetivo inferir el valor de un aspecto de contexto de alto nivel en base a la historia.

A primera vista, las condiciones generales de una aplicación del algoritmo son similares a las de un problema típico de data mining. Sin embargo, existen diferencias importantes que tornan inaplicables o ineficientes ciertos procesos y algoritmos de data mining en el ambiente de aplicaciones sensibles al contexto.

La tarea de clasificación en un problema típico de data mining consta de las siguientes etapas [Orallo 2004d]:

- Pre-procesamiento: en esta etapa se asume que se cuenta con una muestra de aprendizaje. Generalmente se realizan análisis estadísticos destinados a:
  - Analizar la representatividad de la muestra: se analiza si la muestra tomada tiene las mismas características estadísticas que la población.
  - Descarte de valores extraños o ejemplos de aprendizaje con valores desconocidos.
  - Rediseño de la muestra: se aplica cuando la muestra contiene un porcentaje excesivamente alto de una clase. En caso que la muestra no sea representativa de la población, se recurre al oversampling, consistente en repetir ejemplos de la clase minoritaria para compensar el sesgo preexistente.
  - Determinación del algoritmo que mejor se adapta al problema. Se dispone de una amplia gama de algoritmos, cada uno tiene sus ventajas y desventajas en relación a un problema determinado.
  - Segmentación de la muestra: la muestra se subdivide en dos partes: aprendizaje y testeo. No existe una regla para determinar la segmentación adecuada, ya que la misma depende del problema y la muestra, aunque el porcentaje utilizado más frecuentemente es 65 % aprendizaje y 35 % testeo. Los ejemplos de testeo sirven para evaluar el modelo construido en el aprendizaje y mejorarlo. Los esfuerzos de mejora se concentran en

la obtención de un modelo mínimo que conserve la capacidad de inferencia.

- Procesamiento: esta etapa consiste en la corrida del algoritmo con distintos valores de los parámetros. El objetivo es efectuar un análisis de hallazgos y sensibilidad.

- Presentación de resultados: consiste en establecer conclusiones basadas en los resultados obtenidos en el procesamiento.

Las condiciones de aplicación de un algoritmo de clasificación, para la determinación de un aspecto de contexto de alto nivel, difieren en los siguientes puntos con un problema típico de data mining.

- Pre-procesamiento: en este punto se observa la mayor diferencia, ya que no se cuenta con una muestra de aprendizaje estática, la misma se obtiene a medida que la aplicación funciona. Por esta razón, no son aplicables los análisis descriptos anteriormente. Sin embargo, se puede realizar un análisis del modelo obtenido luego de cierta cantidad de ejemplos de aprendizaje. El algoritmo descripto en esta tesis efectúa una reestructuración de la estructura luego de una cantidad de ejemplos ingresada como parámetro.

- Procesamiento: en este contexto el algoritmo infiere y aprende de a un ejemplo por vez (interacción con el usuario). La diferencia está dada por el carácter "on line" que tiene el tipo de problemas de interés en esta tesis.

8.3 Descripción del problema

Se desea diseñar una aplicación para teléfonos móviles que efectúe automáticamente las configuraciones necesarias, de manera tal que el uso del aparato sea más confortable para el usuario.

El teléfono móvil se ha convertido en un aparato de uso cotidiano, su utilización permite estar comunicado en cualquier momento y lugar, sin embargo muchas veces se convierte en una molestia tanto para el propio usuario como para los demás. La idea del ejemplo es simplificar las configuraciones rutinarias que debe realizar el usuario para mejorar el uso del mismo. Los servicios que se activarán dependen de los valores de dos aspectos de contexto, uno determinístico y otro no determinístico. La asignación de servicios se realiza en base a reglas determinísticas, ya que no existe incertidumbre sobre la adaptación necesaria en cada caso.

Servicios

Los servicios disponibles para el usuario son los siguientes:

- Derivación de llamada a un teléfono alternativo (laboral).
- Nivel de sonido, activación de vibrador y activación de luces.
- Activación sistema telefónico automóvil.

- Activación del contestador automático.

Aspectos de contexto a los cuales reaccionará la aplicación

Locación del usuario: el valor de este aspecto de contexto se determinará mediante sensores de proximidad. Este es un aspecto de contexto del tipo determinístico. Las locaciones posibles son las siguientes:

- Automóvil: se considera que el usuario está en el automóvil y conduciendo.
- Trabajo: se considera que el usuario está en su oficina.
- Casa: se considera que el usuario está en cualquier parte de su casa.
- Otros: se considera que el usuario está en algún lugar desconocido, el cuál no se puede establecer por falta de sensores.

Configuración deseada: este aspecto de contexto es del tipo derivado no determinístico. El valor de este aspecto de contexto se infiere en base al valor de los siguientes aspectos de contexto atómicos:

- Luminosidad: este aspecto de contexto es de tipo continuo y se discretizará en dos términos lingüísticos (Alta y Baja).
- Intensidad de ruido: este aspecto de contexto es de tipo continuo y se discretizará en dos términos lingüísticos (Alta y Baja).
- Momento del día: este aspecto de contexto es de tipo continuo y se discretizará en dos términos lingüísticos (Diurno y Nocturno).

Dado que la relación entre los aspectos de contexto atómicos y el aspecto de contexto derivado es desconocida, el valor de este último se debe inferir basándose en la historia. Los valores que puede tomar este aspecto de contexto son: {Discreta, Normal}.

Restricciones para brindar los servicios al usuario

En el framework Ballon cada proveedor de servicio se configura con una restricción (que puede ser compuesta) y uno o varios servicios. Los proveedores de servicios configurados quedarían de la siguiente manera:

ProveedorServicio1

Servicios: Activar contestador automático, desactivar sonido, activar luz y vibrador.
Restricción: Locación = “Casa” y Configuración = “Discreta”. Esta regla indica que el usuario está descansando y no desea ser molestado.

ProveedorServicio2

Servicio: Activar servicio de derivación de llamada, desactivar sonido, activar luz y vibrador.
Restricción: Locación = “Trabajo” y Configuración = “Discreta”. Esta regla indica que el usuario está ocupado, por lo tanto decide derivar las llamadas.

ProveedorServicio3

Servicio: Activar sonido nivel medio.
Restricción: Locación = {"Casa", "Trabajo"} y Configuración = "Normal". Esta regla indica que se debe restaurar la configuración normal.

ProveedorServicio4

Servicio: Activación sistema telefónico automóvil.
Restricción: Locación = "Automóvil". Esta regla asume que el usuario está en un su automóvil y no puede distraerse para tomar el teléfono.

ProveedorServicio5

Servicio: Desactivar sonido, activar luz y vibrador.
Restricción: Locación = "Otros" y Configuración = "Discreta". Esta regla indica que el usuario no desea ruidos molestos para las personas cercanas.

ProveedorServicio6

Servicio: Activar sonido nivel bajo, activar luz y vibrador.
Restricción: Locación = "Otros" y Configuración = "Normal". Esta regla indica que el usuario desea sonido, pero con un tono inferior al normal.

8.4 Análisis referentes al algoritmo

8.4.1 Casos especiales

Atributo sin valor

Este es un problema que puede ocurrir con frecuencia cuando los datos se capturan por medio de sensores. Generalmente esta clase de problemas se soluciona antes de llegar al algoritmo (implementación del Dispatcher). En caso que se deba tratar en el algoritmo, se puede tomar alguna de las siguientes decisiones:

- Desechar el ejemplo: tomando esta decisión se puede perder información valiosa presente en los otros atributos del ejemplo.
- Asignarle un valor: consiste en asignarle un valor antes de la fuzzyficación. La asignación puede hacerse aplicando la moda o el promedio de los valores guardados.
- Asignar grados de membresía: consiste en asignarle un grado de membresía uniforme a todos los términos fuzzy. Esta decisión es acorde a la métrica utilizada para seleccionar el atributo que mejor separa los ejemplos de aprendizaje.

Valores fuera de límite

En el momento de definición del atributo se declaran los valores frontera. El dispatcher (antes de la llegada al fuzzificador) deberá rechazar un valor fuera de límites.

Atributos irrelevantes

Los atributos que presentan el mismo valor en toda la muestra tienen valor predictivo nulo. El mismo caso sucede con los atributos cuyos valores son todos distintos. La utilidad del atributo para separar los ejemplos de aprendizaje en particiones puras (pertenecientes a la misma clase) es nula, por lo tanto son irrelevantes.

El comportamiento adecuado de un algoritmo clasificador es desechar este tipo de atributos. Dadas las condiciones de parada del algoritmo, el atributo sólo puede ser elegido cuando no haya alternativas. La elección como atributo ganador en el proceso de selección, produciría restricciones redundantes.

Muestra sesgada

La muestra está sesgada cuando la mayoría de los ejemplos de aprendizaje suministrados pertenecen a la misma clase. La pureza mínima requerida (criterio de parada del algoritmo) se cumple rápidamente, por lo tanto se generan pocas reglas. Este es el comportamiento adecuado en un algoritmo de clasificación, asumiendo que la muestra es representativa. El concepto de modelo mínimo indica que se deben generar reglas específicas sólo cuando la precisión de inferencia sea mayor a la obtenida con el clasificador más sencillo posible. Este último consiste en asignar a cada ejemplo la clase más frecuente de la muestra.

8.4.2 Costo de creación de un nodo

La eficiencia del algoritmo incremental radica en el almacenamiento de los conteos de ejemplos. En cada nodo se necesita llenar tantas matrices como atributos de inferencia remanentes haya. Cada matriz tiene tantas filas como términos fuzzy tenga el atributo de inferencia y tantas columnas como clases. Cada elemento de la matriz representa la cantidad de ejemplos que satisfacen las condiciones acumuladas hasta el nodo, agregándole la condición dada por el término fuzzy de la fila y la clase de la columna. El almacenamiento de estos conteos facilita el cálculo de la heurística de selección de atributos y las métricas de calidad de las reglas cuando el nodo es hoja.

La cantidad de matrices a llenar disminuye cuando aumenta la profundidad del nodo, dado que disminuye la cantidad de atributos remanentes. Por otra parte las condiciones acumuladas son más largas, esto hace más lento el cálculo de la contribución de cada ejemplo.

El diseño del nodo hace eficiente la actualización de los conteos cuando se agrega un ejemplo de aprendizaje, pero como contrapartida la creación de un nodo es más costosa que en un algoritmo no incremental. Por lo tanto, el algoritmo será eficiente cuando el agregado de un ejemplo no provoque la creación de un nodo. El algoritmo no creará nodos si el ejemplo ingresado es consistente con la estructura. Para el cálculo de los conteos necesarios en un nodo nuevo se necesita tener en cuenta todos los ejemplos de aprendizaje, no se puede reutilizar los conteos almacenados en el nodo padre.

A continuación se describen las variables contenidas en cada nodo.

Condición nodo: almacena la restricción correspondiente al nodo.

Condiciones acumuladas: almacena las restricciones acumuladas desde la raíz hasta el nodo.

Atributos remanentes: almacena los atributos que todavía no se utilizaron en el camino desde la raíz hasta el nodo.

Valores: esta variable almacena las matrices que contienen los conteos.

Distribución: almacena la cantidad de ejemplos pertenecientes a cada clase.

Tamaño partición: almacena la cantidad de ejemplos que pertenecen al nodo.

Atributo discriminador: almacena el atributo seleccionado para discriminar los ejemplos contenidos en el nodo.

Información promedio: almacena el valor de la información promedio correspondiente al atributo discriminador.

La confianza de la regla se puede calcular con las variables Distribución y Tamaño Partición. El soporte es el Tamaño Partición sobre la cantidad de ejemplos totales.

8.4.3 Análisis del aprendizaje

El algoritmo descripto en esta tesis es incremental, por lo tanto es importante conocer el desempeño del mismo a medida que se agregan ejemplos de aprendizaje. En este tipo de captura de datos es normal que el proceso pase del estado maduro a inmaduro o viceversa rápidamente, por este motivo el énfasis está puesto en el análisis de casos. Para caracterizar el proceso de aprendizaje utilizaremos las métricas definidas en el capítulo dedicado al algoritmo.

En los experimentos se utilizará el conjunto de datos mencionados en el *Anexo A*. Para simplificar la nomenclatura y dado que el aspecto de contexto no determinístico tiene dos valores posibles, tomaremos la siguiente convención: "Pos" significa "Normal" y "Neg" significa "Discreto".

Porcentaje hojas creadas = Cantidad hojas creadas / Cantidad hojas posibles
Promedio restricciones = Promedio de restricciones en las hojas creadas.
Porcentaje hojas puras = Cantidad hojas puras / Cantidad hojas creadas.

Cantidad hojas posibles = 2 * 2 * 2 = 8, dado que cada variable lingüística (atributo de inferencia) tiene dos términos fuzzy asociados.

Caso muestra inicial pura

Parámetros

Pureza mínima: 70%
Cantidad mínima de ejemplos relevantes: 1,5
Intervalo de optimización: 5 ejemplos

Luego de ingresar los primeros 10 ejemplos de aprendizaje no fue necesario subdividir el nodo raíz del árbol, dado que la partición cumple con la pureza mínima (pureza = 100%). Esto significa que la base de reglas tendrá una sola regla, la misma tiene un antecedente nulo (sin restricciones) y como consecuente la clase más frecuente.

Cantidad hojas puras = 1
Porcentaje hojas creadas = 1/8 = 12,5 %
Promedio restricciones = 0
Porcentaje hojas puras = 1/1 = 100%

Las métricas indican un proceso de aprendizaje óptimo.

Caso muestra equilibrada

El objetivo del experimento es medir como se comporta el algoritmo de optimización, con distinto niveles de pureza como parámetro, sabiendo que la muestra es equilibrada. Para este caso se mostrará la estructura obtenida luego de ingresar los primeros 20 ejemplos de aprendizaje.

Cantidad mínima de ejemplos relevantes: 1,5
Porcentaje hojas creadas = %Hojas
Promedio restricciones = PR
Porcentaje hojas puras = % Puras

Las métricas obtenidas antes de la optimización son los siguientes:

Pureza	Hojas	% Hojas	Hojas Puras	% Puras	PR
0,7	5	0,625	2	0,4	2,6
0,9	8	1	0	0	3

Las métricas obtenidas luego de la optimización son los siguientes:

Pureza	Hojas	% Hojas	Hojas Puras	% Puros	PR
0,7	4	0,5	2	0,5	2,25
0,9	8	1	0	0	3

El cuadro muestra una mejora en las métricas de aprendizaje luego de la optimización cuando la pureza es 0,7. El algoritmo de optimización podó una ramificación heredada (de la optimización anterior), bajando la cantidad de hojas generadas y aumentando el porcentaje de hojas puras.

Cuando la pureza exigida es 0,9 las métricas no cambian. La no existencia de hojas puras indica que el algoritmo no paró la subdivisión por pureza. Esto significa que la muestra es demasiado inconsistente para soportar el nivel de pureza requerido, obligando al algoritmo de aprendizaje a crear todas las reglas posibles. El algoritmo de optimización no puede mejorar las métricas dado que la estructura heredada es óptima para el nivel de pureza pedido.

Caso muestra sesgada

El objetivo del experimento es medir como se comporta el algoritmo de optimización, con distinto niveles de pureza como parámetro, sabiendo que la muestra está sesgada. Para este caso se mostrará la estructura obtenida luego de ingresar los primeros 30 ejemplos de aprendizaje.

Cantidad mínima de ejemplos relevantes: 1,5
Porcentaje hojas creadas = %Hojas
Promedio restricciones = PR
Porcentaje hojas puras = % Puras

Las métricas obtenidas antes de la optimización son los siguientes:

Pureza	Hojas	% Hojas	Hojas Puras	% Puros	PR
0,7	5	0,625	2	0,4	2,6
0,9	8	1	0	0	3

Las métricas obtenidas luego de la optimización son los siguientes:

Pureza	Hojas	% Hojas	Hojas Puras	% Puros	PR
0,7	4	0,5	2	0,5	2,25
0,9	8	1	0	0	3

Cuando la pureza exigida es 0,7 las métricas de aprendizaje muestran una mejora. Luego de realizar el experimento se observa que el algoritmo de optimización cumplió sus dos misiones: asegurar que las decisiones en cada nodo sea la indicada por la heurística y las condiciones de parada sólo se cumplan en las hojas. La optimización reconstruyó la estructura heredada provocando una mejora en el porcentaje de hojas puras.

Cuando la pureza exigida es 0,9 las métricas no cambian. Las métricas obtenidas luego de la optimización indican que la muestra es demasiado inconsistente para la pureza requerida.

8.4.3.1 Conclusiones referidas al aprendizaje

Los experimentos muestran casos relevantes que se pueden presentar, distinguiéndose los siguientes puntos interesantes:

- Efectividad del algoritmo de optimización: se mostró que el algoritmo cumple con sus dos objetivos: asegurar que las condiciones de parada se cumplan sólo en las hojas y asegurar que cada subdivisión coincida con la heurística de selección. Cuando la muestra es consistente produce una mejora importante en las métricas de aprendizaje.
- Utilidad de las métricas: se mostró la habilidad de las métricas definidas para capturar el desempeño del aprendizaje. Estas métricas pueden aplicarse a cualquier algoritmo de aprendizaje que genere reglas.
- Inestabilidad del modelo: esto se puso de manifiesto en el cambio estructural que puede ocasionar un sólo ejemplo. La cantidad de hojas generadas aumenta notoriamente cuando el nivel de pureza exigido es

superior a la consistencia de la muestra. La explicación de este fenómeno es la inconsistencia que puede provocar el empleo de lógica difusa.

- Las decisiones hechas en los niveles cercanos a la raíz son de mejor calidad: la heurística de selección (información promedio) generalmente aumenta a medida que el algoritmo se acerca a las hojas de la estructura, produciendo particiones de baja calidad (similar pureza que el nodo padre). La heurística tiene en cuenta el tamaño de las particiones creadas, ya que tiende a seleccionar atributos que separen los ejemplos de aprendizaje en particiones puras y tamaños equilibrados.
- El proceso de aprendizaje no mejora necesariamente con la cantidad de ejemplos: el experimento muestra el pasaje de una muestra consistente a inconsistente y viceversa a medida que aumenta la cantidad de ejemplos.

El último punto señalado se corresponde con un cambio de patrón de comportamiento por parte del usuario. El algoritmo tomará dicho cambio como una inconsistencia reflejada en el empeoramiento de las métricas. Este comportamiento se puede consolidar o no.

8.4.3.2 Inteligibilidad de las reglas creadas

Dado que uno de los objetivos planteados en la tesis es obtener un conjunto de reglas adecuado para el usuario, se analizará el desempeño del algoritmo según los aspectos que afectan la inteligibilidad de las mismas.

Redundancia: el algoritmo incurre en una redundancia cuando la clase predominante es la misma en todos los nodos hojas de un mismo padre (ver ejemplo redundancia). Esta redundancia eleva innecesariamente la cantidad de reglas obtenidas.

Ejemplo redundancia

Luz = Alto AND Sonido = Alto → P
Luz = Alto AND Sonido = Bajo → P

Siendo Alto y Bajo los términos fuzzy posibles de la variable lingüística Sonido.

Cantidad de restricciones en el antecedente de las reglas: las reglas se obtienen sólo de las hojas, por esta razón no es válida la existencia de una regla cuyo antecedente esté incluido completamente en el antecedente de otra regla. Por ejemplo:

Luz = Alto AND Momento = Diurno AND Sonido = Alto → P
Luz = Alto AND Momento = Diurno → P
Luz = Alto → P

La existencia de una regla invalida las otras.

Inconsistencia: el algoritmo sólo mostrará la clase dominante como consecuente de la regla, por lo tanto este problema está descartado.

Cantidad de reglas: dado que un atributo no puede participar más de una vez en las restricciones que llegan a una hoja, la cantidad de reglas está acotada. Si tomamos

como inalterables las funciones de membresía de los atributos, la cantidad de reglas depende del grado de inconsistencia en la muestra y el parámetro de pureza fijado.

8.4.4 Análisis de la inferencia

Para estudiar el algoritmo de inferencia se harán experimentos sobre casos relevantes. Suponemos que el algoritmo de aprendizaje es el estudiado en esta tesis. En todos los casos se estudiará el comportamiento de inferencia variando sólo la pureza mínima requerida, dado que es el parámetro más importante. El método de inferencia utilizado es el Average Weigth, dado que arroja un resultado de inferencia con grado de correspondencia.

Caso muestra sesgada

El objetivo del experimento es medir la capacidad de inferir la misma clase que la aprendida, en el caso de tener una muestra sesgada. Para el experimento se tomará el mismo conjunto de aprendizaje.

Nivel de pureza 0,7

	Clasif		
	POS	**NEG**	
POS	21	0	**Real**
NEG	8	1	
Correcion	0,73		
Incorreccion	0,27		
Precision POS	0,72		
Alcance POS	1		
Precision NEG	1		
Alcance NEG	0,11		

Nivel de pureza 0,9

	Clasif		
	POS	**NEG**	
POS	21	0	**Real**
NEG	8	1	
Correcion	0,73		
Incorreccion	0,27		
Precision POS	0,72		
Alcance POS	1		
Precision NEG	1		
Alcance NEG	0,11		

Los cuadros muestran que el algoritmo tiende a inferir la clase dominante cualquiera sea el nivel de pureza requerido. La precisión de la clase dominante es baja por la cantidad de falsos positivos. El alcance máximo de la clase dominante indica que el algoritmo clasificó correctamente todos los ejemplos que pertenecen realmente a la clase. El porcentaje de ejemplos correctos es apenas superior al que se obtendría con el clasificador más sencillo posible (0,7). Este último simplemente clasifica un ejemplo según la clase dominante.

Caso muestra equilibrada

El objetivo del experimento es medir la capacidad de inferir la misma clase que la aprendida, en el caso de tener una muestra equilibrada. Para el experimento se tomará el mismo conjunto de aprendizaje.

Nivel de pureza 0,7

	Clasif		
	POS	**NEG**	
POS	10	1	**Real**
NEG	1	8	
Correcion	0,9		
Incorreccion	0,1		
Precision POS	0,91		
Alcance POS	0,91		
Precision NEG	0,89		
Alcance NEG	0,89		

Nivel de pureza 0,9

	Clasif		
	POS	**NEG**	
POS	10	1	**Real**
NEG	3	6	
Correcion	0,8		
Incorreccion	0,2		
Precision POS	0,77		
Alcance POS	0,91		
Precision NEG	0,86		
Alcance NEG	0,67		

El análisis de los cuadros indica que el algoritmo de inferencia se comporta mejor utilizando la estructura construida con menor nivel de pureza. El mayor nivel de pureza provoca un mayor apego a la inconsistencia de la muestra, provocando menor calidad en las métricas de inferencia.

Este experimento muestra que al aumentar el nivel de pureza se pueden obtener peores métricas de inferencia.

Comparación estructura optimizada y sin optimizar

El objetivo de este experimento es conocer el comportamiento de la inferencia antes y después de optimizar. El experimento es interesante dado que el algoritmo deberá inferir con la estructura heredada entre una optimización y otra. El conjunto de ejemplos de inferencia es el mismo que el conjunto de aprendizaje.

Nivel de pureza 0,7 antes de optimizar

	Clasif		
	POS	**NEG**	
POS	10	1	**Real**
NEG	3	6	
Correcion	0,8		
Incorreccion	0,2		
Precision POS	0,77		
Alcance POS	0,91		
Precision NEG	0,86		
Alcance NEG	0,67		

Nivel de pureza 0,7 luego de optimizar

	Clasif		
	POS	**NEG**	
POS	10	1	**Real**
NEG	1	8	
Correcion	0,9		
Incorreccion	0,1		
Precision POS	0,91		
Alcance POS	0,91		
Precision NEG	0,89		
Alcance NEG	0,89		

El análisis de los cuadros muestra que la optimización mejora las métricas de inferencia, ya que la estructura es más general. Las métricas de inferencia obtenidas con el mayor nivel de pureza (0,9) son iguales antes y después de la optimización. En este último caso la estructura permaneció sin cambios antes y después de optimizar.

Inferencia con ejemplos no aprendidos

El objetivo del experimento es conocer el comportamiento del algoritmo de inferencia, al clasificar una muestra que contenga una mezcla de ejemplos de aprendizaje y ejemplos externos a la misma. Este es el contexto normal en el cual se desempeñará el algoritmo. Para desarrollar este experimento se tomaron los primeros 20 ejemplos de aprendizaje. La muestra de aprendizaje es equilibrada. La estructura obtenida con el nivel de pureza 0,9 es la máxima posible (reflejado en las pésimas métricas de aprendizaje), en cambio es aceptable con nivel de pureza 0,7.

Nivel de pureza 0,7

	Clasif		
	POS	**NEG**	
POS	3	8	**Real**
NEG	1	8	
Correcion	0,55		
Incorreccion	0,45		
Precision POS	0,75		
Alcance POS	0,27		
Precision NEG	0,5		
Alcance NEG	0,89		

Nivel de pureza 0,9

	Clasif		
	POS	**NEG**	
POS	4	7	**Real**
NEG	3	6	
Correcion	0,5		
Incorreccion	0,5		
Precision POS	0,57		
Alcance POS	0,36		
Precision NEG	0,46		
Alcance NEG	0,67		

Las métricas reflejan un empeoramiento significativo respecto a la inferencia obtenida con el mismo conjunto de aprendizaje. Sin embargo, el análisis de los cuadros indica que se obtuvo mejores resultados con el nivel de pureza menor.

8.4.5 Guía para establecer los parámetros

En este tipo de aplicaciones la muestra de aprendizaje se construye mientras la aplicación funciona, por lo tanto no se conoce a priori la representatividad de la misma. Analizaremos cada uno de los parámetros para brindar una guía.

Pureza mínima: esta debería ser la principal condición de parada del algoritmo de aprendizaje. El valor debe estar incluido en el rango [1/|C|, 1], siendo |C| la cantidad de clases. Cuanto más alto sea este valor, mayor será la exposición del modelo a la inconsistencia. Si la muestra es altamente inconsistente y la pureza mínima es alta, el algoritmo creará un gran número de reglas específicas, este resultado se refleja en las métricas porcentaje de nodos creados y cantidad promedio de restricciones. Por lo visto en los experimentos, es conveniente utilizar un nivel de pureza bajo, ya que es más seguro obtener una buena estructura de aprendizaje y buen desempeño en inferencia.

Cantidad mínima de ejemplos relevantes: este valor representa la cantidad mínima de ejemplos necesarios para avalar el agregado de una nueva restricción. El valor debería ser mayor que 1, dado que no es deseable subdividir un nodo cuyo tamaño

sea tan sólo un ejemplo. Es conveniente establecer este parámetro en un valor bajo para que no tenga incidencia cuando se tienen pocos ejemplos de aprendizaje.

Intervalo de optimización: la frecuencia de optimización ideal depende del estado del aprendizaje. Si el algoritmo crea gran cantidad de nodos y/o se obtiene un bajo porcentaje de hojas puras respecto de la optimización anterior, existen fuertes indicios de la necesidad de optimizar. A diferencia de los parámetros de pureza y cantidad mínima de ejemplos, este valor se puede modificar durante la ejecución de la aplicación.

8.5 Análisis referentes a la implementación

8.5.1 Momentos claves

Cuando se implementa una aplicación de este tipo se debe determinar claramente ciertas cuestiones claves, estas son los siguientes:

Toma del ejemplo de aprendizaje: suponiendo que los valores de los aspectos de contexto de inferencia están disponibles, se necesita adjuntarle una clase para obtener un ejemplo de aprendizaje. Existen dos modos, los cuales no son excluyentes, para realizar el trabajo.

- Pregunta directa al usuario: esta es la mejor de todas, pero puede resultar la más molesta para el mismo. La aplicación puede tornarse inservible si el valor que aporta es menor al esfuerzo necesario para mantenerla. Por esta razón es recomendable utilizarla sólo en los primeros ejemplos de aprendizaje. Esta opción se implementa en una aplicación con una simple ventana de diálogo que informe el valor de los aspectos de contexto y la clase inferida, el usuario deberá responder la inferencia correcta o descartar el ejemplo por considerarlo una anomalía
- Asumir el valor según una convención: esta opción tiene la ventaja de no molestar al usuario, pero presenta una incertidumbre. En el presente ejemplo, podemos suponer la clase cuando el usuario realiza una acción de configuración.

Momento de aprender: este momento se refiere al instante en el cual el algoritmo de aprendizaje tiene en cuenta los nuevos ejemplos disponibles. El momento ideal es luego de conocer el valor correcto de la clase. De esta manera se mantiene actualizado el comportamiento del usuario.

Momento de inferencia: el modelo descripto en esta tesis se comporta como un sensor de alto nivel. Los sensores de menor nivel disparan una advertencia cuando existe un cambio significativo en el valor sensado. El disparo de algún sensor de bajo nivel propaga el cambio, por lo tanto la necesidad de inferencia. Este proceso depende también de la política del sensor, esta puede ser Push, en este caso el sensor toma un rol pasivo o Pull, en tal caso el sensor toma un rol activo. En este último caso se debe configurar el inicio de toma de valores y el intervalo de repetición.

Momento de entregar los servicios: este momento se refiere al instante en el cual se evalúan las restricciones de los distintos proveedores de servicios. Este debe ser cuando cambia el valor de un aspecto de contexto. En el caso del aspecto derivado, luego de realizar la inferencia. Las reglas presentes en los proveedores de servicios son determinísticas y las fija el usuario con antelación.

8.5.2 Redundancia de información

El conocimiento almacenado en el algoritmo de aprendizaje y la base de reglas significan una redundancia de información. Esta es una condición de diseño que cumple los siguientes propósitos:

Permitir la actualización del usuario: cuando los patrones de comportamiento aprendidos en forma automática no reflejan la respuesta deseada del usuario, este la puede cambiar, por ejemplo manipulando los grados de certeza de las reglas. El usuario puede interrumpir el refresco automático de las reglas o reanudarlo en cualquier momento. Generalmente, el usuario deseará intervenir en el proceso de maduración.

Cumplir el rol de back-up: la inferencia se realiza por medio de la base de reglas, por lo tanto esta debe estar disponible la mayor parte del tiempo posible. El algoritmo de aprendizaje tiene una demora que puede ser demasiado alta, en especial cuando se corre el algoritmo de optimización.

Permitir la utilización de distintos algoritmos de aprendizaje e inferencia: la base de reglas es el punto de convergencia para los algoritmos que generen reglas de comportamiento, por ejemplo el a priori modificado para clasificación. Luego de obtener la base de reglas se pueden aplicar a la misma distintos algoritmos de inferencia y poda.

8.5.3 Limitaciones de hardware

Las aplicaciones sensibles al contexto se implementan usualmente en dispositivos móviles. Estos tienen una capacidad limitada de memoria y procesamiento. El algoritmo incremental trata de resolver dichas limitaciones (ver costo de creación de un nodo).

El número de objetos creados durante la implantación del algoritmo está acotado por la cantidad de reglas posibles, esta última depende de la cantidad de atributos de inferencia y los términos lingüísticos definidos para cada uno.

El repositorio de casos es el único objeto con crecimiento ilimitado. Para disminuir este problema se diseñó como una entidad separada que presta servicios de conteo al algoritmo de aprendizaje, de esta manera se facilita la implementación en un hardware separado. Aunque se utilice un algoritmo incremental, el uso de lógica difusa en el algoritmo de aprendizaje implica la necesidad de tomar todos los ejemplos guardados (ver costo de creación de un nodo). Por esta razón el tiempo de respuesta del algoritmo de aprendizaje depende en gran parte de la implementación eficiente del repositorio de casos.

8.6 Ámbito de aplicación del modelo

El modelo descripto en esta tesis está destinado a aspectos de contextos derivados referidos a gustos, costumbres, estados de ánimo del usuario, etc. Los rasgos que caracterizan a estos aspectos de contexto son:

- El aprendizaje se realiza a medida que el usuario interactúa con la aplicación. Por ejemplo, no se puede conocer de antemano el gusto de un usuario.
- El comportamiento del usuario puede cambiar a medida que transcurre el tiempo.
- Se conocen los aspectos de contexto atómicos necesarios para la inferencia, pero no se conocen las reglas de interacción entre ellos y el aspecto de contexto derivado.
- La clase es nominal: en términos de lógica difusa, cada clase tendrá un grado membresía 0 o 1.

Los atributos de inferencia pueden ser discretos o continuos. Estos últimos se deben discretizar antes de la aplicación del algoritmo. La utilización de particiones fuzzy es una solución a este problema.

Capítulo IX Conclusión y Trabajos futuros

9.1 Conclusión

El software descripto en esta tesis tiene como objetivo la inferencia de aspectos de contexto derivados no determinísticos, basándose en la historia de uso. En el capítulo de arquitectura se describen los objetos integrantes del mismo y la misión de cada uno. El modelo desarrollado permite la incorporación de distintos algoritmos de aprendizaje e inferencia. Esto se logró mediante la creación de una base de reglas, objeto en cual confluyen los distintos algoritmos que generen reglas de comportamiento en forma automática.

En el capítulo destinado a la conexión con el framework Ballon se describe la manera de reutilizar el tratamiento de los sensores y la capa de servicios, considerados puntos fuertes del mismo. Para mejorar la adaptabilidad del software propuesto, se modeló como un sensor de alto nivel, el cual infiere valores en vez de medir directamente un aspecto de contexto, tal como lo haría un sensor tradicional.

El capítulo dedicado al algoritmo describe uno de los algoritmos adecuados para lograr el objetivo planteado. El mismo tiene como características más importantes: la incorporación de lógica difusa y captura de muestra de aprendizaje incremental. Dichas características permiten la actualización dinámica de las reglas de comportamiento y una comunicación amigable para el usuario.

Finalmente, en el capítulo dedicado al ejemplo, se muestra la idea de combinación con un aspecto de contexto determinístico. De esta manera la aplicación resultante puede ser sensible a aspectos de contexto determinísticos y no determinísticos en forma transparente. Esta posibilidad representa un alto valor agregado para el diseñador de una aplicación sensible al contexto. En dicho capítulo se analizó el algoritmo, haciendo hincapié en el estudio de casos, y los problemas vinculados a la implementación del modelo descripto en esta tesis.

9.2 Trabajos futuros

Mejoras al algoritmo

- Reducir la redundancia producida por el algoritmo (ver análisis del aprendizaje). Una de las maneras más eficientes sería utilizando la información promedio, ya que en cada nodo se guarda su valor. Si la información en el nodo es alta, significa que el algoritmo está forzado a conseguir una pureza que la muestra no tiene, lo cual se traduce en particiones innecesarias. Una heurística que se podría tomar es la siguiente: si la información del nodo aumenta significativamente, indica que la partición del nodo es innecesaria. Siguiendo esta heurística la partición no se debería realizar, aunque no se cumplan las condiciones de parada tradicionales del algoritmo de aprendizaje.
- Cambiar dinámicamente el parámetro “frecuencia optimizar”: la necesidad de optimizar, depende de la consistencia del conjunto de ejemplos de

aprendizaje suministrado. En un dominio altamente cambiante, la frecuencia de optimización debe ser mayor que en un dominio estable. El algoritmo podría detectar esta situación basándose en la cantidad de reestructuraciones producidas en cada proceso de optimización. Las métricas de aprendizaje definidas pueden ser de gran ayuda. Este parámetro también podría basarse en tiempo. El algoritmo de optimización, gran consumidor de recursos, podría ejecutarse en horas de baja actividad de inferencia.

- Costo de equivocación: existen situaciones en las cuales la equivocación por una clase tiene distinta connotación que otra. Dicho sesgo se podría incluir en el algoritmo para que la inferencia se realice con mayor precisión, evitando de esta manera situaciones molestas para el usuario.

Referencias

[Acid 1997] An algorithm for learning probabilistic belief networks. Silvia Acid, Luis Campos. Universidad de Granada, España.

[Agrawal 1993] Mining association rules between sets of items in large databases. Rakesh Agrawal, Tomasz Imielinski, Arun Swami. IBM Almaden research Center 1993.

[Alpert 1998] The design patterns Smalltalk companion. Sherman Alpert, Kyle Brown, Bobby Woolf Addison Wesley 1998.

[Au 2001] Classification with degree of membership: a fuzzy approach. Wai-Ho Au, Keith Chan. Department of computing, The Hong Kong Polytechnic University. 2001.

[Borisov 2003] Construction of incremental fuzzy decision trees. Arkady Borisov, Gulnara Bikesheva. Riga Technical University. 2003.

[Bayardo 1999] Mining the most interesting rules. Roberto Bayardo, Rakesh Agrawal. IBM Almaden research Center 1999.

[Cheng 1998] Comparing Bayesian network classifiers. Jie Cheng, Russel Greiner. University of Alberta, Canada.

[Cheverst 2003] Supporting proactive "intelligent" behaviour: the problem of uncertainty. Lancaster university, United Kindom.

[Dubois 2003] A note on quality measures for fuzzy association rules. Didier Dubois, Henri Prade. Instituto de investigación en informática de Toulouse, Francia. 2003.

[Fahy 2004] Cass-Middleware for mobile context-aware applications. Patrick Fahy Siobhan Clarke. Computer science department, Trinity College Dublin, Ireland.

[Fernández 2004] Análisis de clasificadores Bayesianos. Trabajo final especialidad Ingeniería en sistemas expertos. Enrique José Fernandez Diciembre 2004. Instituto Tecnológico de Buenos Aires.

[Fiszelew 2001] Generación automática de redes neuronales con ajuste de parámetros basados en algoritmos genéticos. Fiscelew A., García Martinez R. Instituto tecnológico de Buenos Aires.

[Fortier 2005] Un enfoque orientado a objetos para software context aware. Tesis de grado UNLP.

[Gamma 1995] Design patterns. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison Wesley 1995.

[Greiner 1998] Learning Bayesian belief network classifiers: Algorithms and system. Jie Cheng, Russel Greiner. University of Alberta.

[Gu 2004] A middleware for building context-aware mobile services. Tao Gu, Hung Keng Pung, Da Qing Zhang. Department of computer science, National University of Singapore. 2004.

[Guetova 2002] Incremental fuzzy decision trees. Marina Guetova, Steffen Holldobler, Hans-Peter Storr. Artificial Intelligence Institute. Department of computer science Dresden University. 2002.

[Ishibuchi 2004] Comparison of heuristic criteria for fuzzy rule selection in classification problems. Hisao Ishibuchi, Takashi Yamamoto. Department of industrial engineering, Osaka Prefecture University. 2004.

[Janikow 1996] Fuzzy decision trees: issues and methods. Cezary Janikow. University of Missouri, St Louis. October 1996.

[Janikow 1994] Exemplar learning in fuzzy decision trees. C. Z. Janikow. Mathematics and computer science, University of Missouri. 1994.

[Korpipaa 2003] Managing context information in mobile devices. Panu Korpipaa, Jani Mantyjarvi, Juha Kela, Heikki Keranen, Esko-Juhani Malm. Technical research center of Finland.

[Kruse 1998] Data mining with graphical models. Rudolf Kruse, Christian Borgelt. University of Magdeburg, Germany.

[Kuok 1996] Mining fuzzy association rules in databases. Chan Man Kuok, Ada Fu, Man Hon Wong. Department of computer science and engineering, Chinese university of Hong Kong. 1996.

[Langseth 2004] Classification using hierarchical Naïve Bayes models. Helge Langseth, Thomas Nielsen. Aalborg university of Denmark.

[Liu 2000] Analyzing the subjective interestingness of association rules. Bing Liu, Wynne Hsu, Shu Chen and Yiming Ma. School of computing National university of Singapore. 2000.

[Marsala 1999] An adaptable system to construct fuzzy decision trees. Christophe Marsala, Bernadette Bouchon-Meunier. M. Curie University, Paris France.

[Marsala 1998] Application of fuzzy rule inductions to data mining. Christophe Marsala. M. Curie University, Paris France.

[Marsala 2000] Fuzzy decision trees to help flexible querying. Christophe Marsala. Universite Pierre et Marie Curie. 2000.

[Megiddo 1998] Discovering predictive association rules. Nimrod Megiddo, Ramakrishnan Srikant. IBM Almaden research Center 1998.

[Mendel 1995] Fuzzy logic systems for engineering: a tutorial. Jerry Mendel. IEEE. 1995.

[Nadkami 1999] A Bayesian network approach to making inferences in causal maps. Sucheta Nadkami, Prakash Shenoy. School of business, University of Kansas.

[Nakashima 2000] Effect of rule weights in fuzzy rule-based classification systems. Hisao Ishibuchi, Tomoharu Nakashima. Department of industrial engineering, Osaka Prefecture University. 2000.

[Orallo 2004] Introducción a la minería de datos. José Orallo, María José Ramirez Quintana, Cesar Ferri Ramirez. Editorial Pearson - Prentice Hall, página 292.

[Orallo 2004b] Introducción a la minería de datos. José Orallo, María José Ramirez Quintana, Cesar Ferri Ramirez. Editorial Pearson - Prentice Hall, página 287.

[Orallo 2004c] Introducción a la minería de datos. José Orallo, María José Ramirez Quintana, Cesar Ferri Ramirez. Editorial Pearson - Prentice Hall, páginas 440-455..

[Orallo 2004d] Introducción a la minería de datos. José Orallo, María José Ramirez Quintana, Cesar Ferri Ramirez. Editorial Pearson - Prentice Hall, páginas 19-39.

[Nath 2004] Fuzzy sets and fuzzy systems. Baikunth Nath. Computer science and software engineering, University of Melbourne. 2004.

[Pan 1998] Fuzzy causal probabilistic networks. Heping Pan, Daniel McMichel. Tecnology park Adelaide, Australia. Mayo 1998.

[Randon 2004] Fuzzy and Random set based induction algorithms. Phd. Tesis University of Bristol.

[Rennie 2004] Tackling the poor assumptions of Naïve Bayes Classifiers. Jason Rennie, Lawrence Shih, Jaime Teevan, David Karger. Artificial Intelligence Lab, MIT. Abril 2004.

[Schiaffino 2003] Using association rules to learn user assistance requirements. Silvia Schiaffino, Analía Amandi. Facultad de ciencias exactas, Universidad Nacional del centro de la provincia de Buenos Aires. ASAI 2003.

[Servente 2002] Algoritmos TDIT aplicados a la minería de datos inteligente. Servente M, García Martinez R. Instituto tecnológico de Buenos Aires.

[Srikant 1996] Mining quantitative association rules in large relational tables. Ramakrishnan Srikant, Rakesh Agrawal. IBM Almaden research Center 1996.

[Storr 2003] A compact fuzzy extension of the Naive Bayesian Classification algorithm. Hans Peter Storr Dresden University Germany.

[Wang 2004] Information measures in fuzzy decision trees. Xiaomeng Wang, Christian Borgelt. Department of computer science, University of Magdeburg. 2004.

[Weka 2004] Weka. The University of Waikato. Framework especializado en algoritmos de data mining versión 3-4, 2002-2004. www.cs.waikato.ac.nz.

[Wong 2000] Mining fuzzy association rules for web access case adaptation. Cody Wong, Simon Shiu, Sankar Pal. Department of computing Hong Kong Polytechnic University, Hong Kong China. 2000.

[Zaiane 2002] Mammography classification by an association rule-based classifier. Osmar Zaiane, Maria Luiza Antonie, Alexandra Coman. Department of computing science, University of Alberta Canadá. 2002.

Anexo A – Ejemplo de utilización del algoritmo

En este anexo se expondrán los ejemplos de aprendizaje utilizados en los experimentos. Adicionalmente, se mostrarán las estructuras intermedias conseguidas.

Atributos de inferencia

Luminosidad: {Alta, Baja}

Función de membresía

X: nivel de luz en Lux. Los valores característicos fueron extraídos del manual de un sensor comercial. (www.baldorsrl.com.ar/informe_manuales/lux/401025.pdf)

Baja

1	x <= 200
(1/(200-800))*(x-200)+1	200 < x <= 800
0	x > 800

Alta

0	x <= 200
(1/(800-200))*(x-200)	200 < x <= 800
1	x > 800

Momento del Día: {Diurno, Nocturno}

Función de membresía

X: horario en minutos y fracción.

Diurno

0	x <= 360
1/(480-360))*(x-360)	360 < x <= 480
1	480 < x <= 1080
(1/(1080-1200))*(x-1080)+1	1080 < x <= 1200
0	1200 < x <= 1440

Nocturno

1	x <= 360
1/(360-480))*(x-360)+1	360 < x <= 480
0	480 < x <= 1080
(1/(1200-1080))*(x-1080)	1080 < x <= 1200
1	1200 < x <= 1440

Sonido: {Alta, Baja}

Función de membresía

X: sonido en Db

Baja

1	x <= 30
(1/(30-70))*(x-30)+1	30 < x <= 70
0	x > 70

Alta

0	x <= 30
(1/(70-30))*(x-30)	30 < x <= 70
1	x > 70

Atributo clase

Configuración deseada: {Discreta, Normal}. Para simplificar la nomenclatura llamaremos "Pos" a la clase "Normal" y "Neg" a la clase "Discreta".

Conjunto de ejemplos

El conjunto de ejemplos mostrado, corresponde a una generación de laboratorio realizada con el fin de analizar el comportamiento de los algoritmos descriptos en esta tesis. Los ejemplos fueron "fuzzyficados" según las funciones de membresía descriptas anteriormente.

Instancia	Luz				Momento del día				Sonido				Configuración			
1	B	0,7	A	0,3	D	0,3	N	0,7	B	0,3	A	0,7	POS	1	Neg	0
2	B	0,6	A	0,4	D	0,4	N	0,6	B	0,4	A	0,6	POS	1	Neg	0
3	B	1	A	0	D	0,2	N	0,8	B	0,2	A	0,8	POS	1	Neg	0
4	B	0,6	A	0,4	D	0,4	N	0,6	B	0,4	A	0,6	POS	1	Neg	0
5	B	0,8	A	0,2	D	0,7	N	0,3	B	0,1	A	0,9	POS	1	Neg	0
6	B	0,7	A	0,3	D	0,8	N	0,2	B	0,3	A	0,7	POS	1	Neg	0
7	B	0,4	A	0,6	D	0,6	N	0,4	B	0,4	A	0,6	POS	1	Neg	0
8	B	0,3	A	0,7	D	0,9	N	0,1	B	0,2	A	0,8	POS	1	Neg	0
9	B	0,2	A	0,8	D	1	N	0	B	0,3	A	0,7	POS	1	Neg	0
10	B	0,1	A	0,9	D	0,8	N	0,2	B	0,1	A	0,9	POS	1	Neg	0
11	B	0,7	A	0,3	D	0,6	N	0,4	B	0,7	A	0,3	POS	0	Neg	1
12	B	0,8	A	0,2	D	0,9	N	0,1	B	0,6	A	0,4	POS	0	Neg	1
13	B	0,6	A	0,4	D	0,4	N	0,6	B	0,8	A	0,2	POS	0	Neg	1
14	B	0,9	A	0,1	D	0,1	N	0,9	B	0,6	A	0,4	POS	0	Neg	1
15	B	1	A	0	D	0	N	1	B	0,9	A	0,1	POS	0	Neg	1
16	B	0,3	A	0,7	D	0,7	N	0,3	B	0,7	A	0,3	POS	0	Neg	1
17	B	0,4	A	0,6	D	0,6	N	0,4	B	0,6	A	0,4	POS	0	Neg	1
18	B	0,2	A	0,8	D	1	N	0	B	0,8	A	0,2	POS	0	Neg	1

19	B	0,4	A	0,6	D	0,6	N	0,4	B	0,7	A	0,3	POS	0	Neg	1
20	B	0,1	A	0,9	D	0,8	N	0,2	B	0,9	A	0,1	POS	1	Neg	0
21	B	0,3	A	0,7	D	0,7	N	0,3	B	0,7	A	0,3	POS	1	Neg	0
22	B	0,4	A	0,6	D	0,9	N	0,1	B	0,6	A	0,4	POS	1	Neg	0
23	B	0,2	A	0,8	D	0,3	N	0,7	B	0,8	A	0,2	POS	1	Neg	0
24	B	0,3	A	0,7	D	0,4	N	0,6	B	0,9	A	0,1	POS	1	Neg	0
25	B	0,1	A	0,9	D	0,2	N	0,8	B	1	A	0	POS	1	Neg	0
26	B	0,3	A	0,7	D	0,4	N	0,6	B	0,8	A	0,2	POS	1	Neg	0
27	B	0,4	A	0,6	D	0,1	N	0,9	B	0,7	A	0,3	POS	1	Neg	0
28	B	0,2	A	0,8	D	0,3	N	0,7	B	0,9	A	0,1	POS	1	Neg	0
29	B	0,1	A	0,9	D	0,4	N	0,6	B	0,7	A	0,3	POS	1	Neg	0
30	B	0	A	1	D	0,2	N	0,8	B	0,6	A	0,4	POS	1	Neg	0

Estructuras significativas obtenidas

Las estructuras se obtuvieron con los siguientes parámetros:

Nivel de pureza mínimo: mayor a 0,7
Cantidad mínima de ejemplos: menor a 1,5
(información promedio)

20 Ejemplos sin optimizar

S = A (0,88)
    Pos: 0,74
S = B (0,92)
    L = B (0,86)
        M = D
            Neg: 0,66
        M = N
            Neg: 0,74
    L = A (0,99)
        M = D
            Neg = 0,55
        M = N
            Neg = 0,58

20 Ejemplos optimizado

S = A (0,88)
    Pos: 0,74
S = B (0,92)
    L = B
        Neg = 0,7
    L = A (0,99)
        M = D
            Neg: 0,55
        M = N
            Neg: 0,58

El árbol con raíz L=B (Luz = Baja) fue podado por el algoritmo de optimización.

30 Ejemplos sin optimizar

S = A (0,86)
    Pos: 0,78
S = B (0,88)
    L = B (0,99)
        M = D
            Neg: 0,51
        M = N
            Neg: 0,55
    L = A (0,77)
        M = D
            Pos: 0,66
        M = N
            Pos: 0,84

30 Ejemplos optimizado

L = A (0,85)
    Pos: 0,78
L = B (0,91)
    S = B (0,99)
        M = D
            Neg: 0,51
        M = N
            Neg: 0,55
    S = A
        Pos: 0,75

La estructura optimizada muestra que el algoritmo decidió cambiar la estructura raíz del árbol, dado que el mejor atributo discriminador ha cambiado. Además, se observa la poda del árbol con raíz S=A (Sonido = Alto). El cambio del atributo discriminador en la raíz, indica que la heurística de selección (información promedio) tiene en cuenta el tamaño de las particiones creadas. El mejor atributo es aquel que divida los ejemplos de aprendizaje en particiones puras y tamaños equilibrados.