UNIVERSIDAD VERACRUZANA

FACULTAD DE FÍSICA E INTELIGENCIA ARTIFICIAL

Una aproximación de aprendizaje colaborativo en sistemas multi-agente con aplicación a minería de datos distribuida

TESIS

QUE PARA OBTENER EL GRADO DE
MAESTRO EN INTELIGENCIA ARTIFICIAL

PRESENTA
HÉCTOR XAVIER LIMÓN RIAÑO

DIRECTOR
DR. ALEJANDRO GUERRA HERNÁNDEZ

CODIRECTOR
DR. NICANDRO CRUZ RAMÍREZ

XALAPA, VER. AGOSTO 2013

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Antecedentes

1.1.1. Agencia BDI

En el llamado nuevo enfoque de la inteligencia artificial (Russell, 2009) los agentes juegan un papel primordial, siendo éstos el principio y finalidad de la inteligencia artificial. Cada agente es un ente situado en un ambiente, el agente puede percibir su ambiente y actuar en consecuencia sobre éste. Un agente racional es aquel que intenta comportarse de la mejor forma posible dado el conocimiento de su ambiente. Los agentes BDI (A. S. Rao y Georgeff, 1991) son una aproximación directa de los principios de agentes racionales, con algunos agregados como son la idea de la integración de un mecanismo intencional (Bratman, 1987) que de algún modo dicta el comportamiento del agente; y de un mecanismo de actos de habla (Searle, 1962), que permite la comunicación entre agentes y abre la puerta a la colaboración y negociación entre éstos. Las siglas BDI representan creencias (believes), deseos (desires) e intenciones (intentions), lo que quiere decir que el agente actúa de acuerdo a lo que conoce, desea o intenta. El mecanismo intencional permite subir de nivel el grado de abstracción en que se definen los programas, cada agente actúa dependiendo de sus creencias, sus deseos y sus intenciones, por lo que el programador debe pensar en esos términos al momento de definir un sistema, esto le permite hacer definiciones intuitivas aún en problemas relativamente complejos. Gran parte del poder de abstracción del mecanismo intencional se debe al hecho de que se utiliza lógica de primer orden para definir el comportamiento del agente, teniendo éste un mecanismo de razonamiento práctico que le permite exhibir un comportamiento complejo.

1.1.2. Jason

En el área de sistemas multi-agente han surgido diversos lenguajes de programación orientados al desarrollo de los mismos. En este caso se centrará la discusión en el lenguaje Jason (Bordini, Hübner, y Wooldridge, 2007). Jason es un lenguaje de programación orientado a agentes basado en Java que implementa una semántica operacional extendida de AgentSpeak. En Jason la definiciones del ambiente se lleva a cabo en términos de Java y el trabajo de razonamiento del agente se lleva a cabo en términos de AgentSpeak, esto permite cierta separación entre ambiente y agentes y brinda la posibilidad de realizar extensiones por medio de bibliotecas Java.

1.1.3. CArtAgO

Una parte fundamental de un sistema Multi-Agente es el ambiente en el que este se despliega. Es importante ser capaz de modelar adecuadamente el ambiente de tal forma que los agentes sean capaces de percibirlo, modificarlo y aprovecharlo. CArtAgO es una infraestructura y arquitectura basada en artefactos utilizado para modelar ambientes de cómputo utilizados en sistemas Multi-Agente. Con CArtAgO el concepto de ambiente es simplificado: el ambiente es un conjunto de artefactos.

Un artefacto es una abstracción de primera clase utilizado para modelar ambientes computacionales en sistemas Multi-Agente (Ricci, Viroli, y Omicini, 2006a). Cada artefacto representa a una entidad del ambiente, brindándole servicios y estructuras a los agentes para que estos mejoren sus actividades, especialmente las sociales. Los artefactos además son de utilidad en el diseño y reutilización de sistemas Multi-Agente pues presentan una estructura modular basada en objetos. Los artefactos están concebidos para ser dispositivos computacionales orientados a la funcionalidad, funcionalidad que los agentes pueden explotar para dar soporte a sus actividades tanto individuales como colectivas (Ricci, Viroli, y Omicini, 2006b). La visión propuesta por CArtAgO impacta directamente en las teorías de agencia sobre interacción y actividad (Ricci y cols., 2006a). Bajo esta visión un sistema Multi-Agente es concebido como un conjunto de agentes que desarrollan sus actividades de tres formas distintas:

- Computando
- Interactuando con otros agentes
- Usando artefactos compartidos que conforman su ambiente computacional

Los artefactos pueden ser tanto el objetivo de la actividad del agente así como las herramientas que los agentes utilizan como medio para cumplir sus actividades, reduciéndose la complejidad de sus tareas. Dado que el ambiente está conformado por artefactos, el estado de cada artefacto puede ser percibido por aquellos agentes para los cuales el artefacto es relevante. La infraestructura de CArtAgO está además pensada para funcionar en ambientes distribuidos, pudiéndose definir espacios de trabajo que determinan el contexto donde un artefacto existe y por lo tanto puede ser percibido y utilizado.

1.1.4. Minería de datos distribuida (DMM)

El descubrimiento de conocimiento en bases de datos, o minería de datos, es una disciplina que conjunta diversas técnicas de exploración y análisis de vastas cantidades de datos con el afán de descubrir patrones y reglas significativas de algún modo ocultas en éstos. Dado que la minería de datos trata con datos, es importante conocer la procedencia y distribución de éstos para determinar la mejor forma de explotarlos de forma efectiva y eficiente. Una forma tradicional de realizar minería de datos es mediante un esquema centralizado, todos los datos y modelos producidos se encuentran en una sola

localidad. Sin embargo, con la actual ubiquidad de las redes de cómputo, es común encontrarse datos pertenecientes a un dominio común que se encuentran distribuidos en diversas localidades. El tener distribución de datos presenta las siguientes problemáticas principales:

- ¿Cuál es la mejor estrategia a seguir para construir modelos de aprendizaje que tomen en consideración los datos de todas las localidades?
- ¿Cómo puede lidiarse con bases de datos heterogéneas?
- Dado que en muchas aplicaciones la totalidad de datos puede ser demasiado grande, ¿Cómo puede optimizarse la comunicación de datos y las operaciones de minería de datos?
- ¿De que forma puede asegurarse la privacidad de los datos transmitidos?
- ¿Cómo pueden tratarse casos de forma eficiente donde los datos de entrada cambian y crecen constantemente?

Muchos sistemas de minería de datos distribuida han sido desarrollados. Estos sistemas pueden clasificarse en tres categorías de acuerdo a la estrategia que implementan (V. S. Rao, 2009):

- Aprendizaje centralizado: Se basa en mover todos los datos a una localidad central, una vez que los todos los datos se encuentran reunidos se integran entre si y se aplican secuencialmente algoritmos de minería de datos. Esta estrategia es usada cuando la totalidad de los datos distribuidos es pequeña. La estrategia es generalmente muy costosa pero más precisa. Para muchos casos esta estrategia no es factible debido a su alto costo. Cabe señalar también que esta estrategia sobre carga a la localidad central, no teniéndose un adecuado balance de carga de trabajo. Esta estrategia tampoco lidia bien con datos heterogéneos, es necesario realizar transformaciones en la localidad central o antes de que los datos sean transmitidos.
- Meta-Aprendizaje: Esta estrategia es usada principalmente en ambientes homogéneos y sigue tres pasos principales. Primero, se generan clasificadores base en cada localidad utilizando algoritmos de construcción de modelos de clasificación. En el segundo paso se reúnen los clasificadores base en una localidad central, y se producen datos de meta-nivel a partir de un conjunto de validación separado que se aplica a los clasificadores base. El tercer paso es generar el clasificador final (meta-clasificador) a partir de los datos de meta-nivel por mediación de un combinador o un arbitro. En general, la precisión de esta estrategia no es tan buena como la de aprendizaje centralizado, sin embargo, su costo es menor, teniéndose menos transmisión de datos y un mejor balance de cargas. Como desventaja, esta estrategia es más difícil de implementar ya que requiere de más pasos intermedios, así mismo, los modelos se aprenden de forma local, pudiéndose tener repercusiones al generalizar. Como ventaja se tiene mayor privacidad de datos dado que sólo los modelos base y quizás los datos de validación son transmitidos.

- Aprendizaje híbrido: Esta estrategia combina aprendizaje local y centralizado para la construcción de modelos, pudiéndose utilizar una u otra estrategia con las mismas ventajas y desventajas. Tanto los modelos como los datos pueden ser transmitidos dependiendo de las necesidades que se tengan. A los sistemas que utilizan esta estrategia se les critica porque no siempre es posible obtener resultados iguales a los obtenidos por una estrategia centralizada. También se les critica debido a que el uso de los recursos de hardware no siempre está optimizado. Esta estrategia, al igual que las demás, no lidia bien con datos heterogéneos.

1.1.5. Minería de datos distribuida basada en agentes

La minería de datos distribuida se aplica en diversos dominios, estos dominios tienen sus propias complicaciones y deben ser tratados de forma especial en cada caso. Las aplicaciones de minería de datos distribuida pueden ser mejoradas mediante agentes, así los agentes se encargaran de lidiar con los detalles de cada caso. El esquema central de trabajo de sistemas multi-agente lo vuelve idóneo para aplicaciones de cómputo distribuido. Cada agente puede realizar diversas tareas concurrentemente y puede ser visto como un proceso independiente cuya localización física es irrelevante a menos que se desee lo contrario. La comunicación entre agentes se realiza a un nivel de abstracción superior que permite transparencia en cuanto a la localización de cada agente dentro de la red de cómputo. La implementación de sistemas distribuidos en términos de sistemas multi-agente es natural y directa ya que el núcleo de estos sistemas es un sistema distribuido en si.

La combinación entre sistemas multi-agente y minería de datos distribuida goza actualmente de mucha popularidad (V. S. Rao, 2009), pudiéndose encontrar herramientas maduras como es el caso de JAM (Stolfo, Prodromidis, y cols., 1997), la cual a servido de influencia a muchos otras herramientas.

1.1.6. Aprendizaje colaborativo de conceptos en sistemas mult-agente

Un problema de aprendizaje de conceptos lidia con mantener consistente una hipótesis acerca de un concepto objetivo, la hipótesis debe ser consistente con un conjunto de ejemplos que pueden ser adquiridos del ambiente o enviados por otros agentes. La hipótesis se mantiene consistente a partir de una serie de revisiones incrementales. En (Bourgne, El Fallah Segrouchni, y Soldano, 2007)la noción de consistencia fue extendida a un grupo de agentes, creando una configuración colaborativa llamada SMILE para el aprendizaje de conceptos. Los resultados obtenidos muestran que un grupo de agentes aprenden mejor un problema booleano complejo que un único agente que recibe todos los ejemplos.

En SMILE los agentes están embebidos en un sistema multi-agente completamente conectado. Cada agente almacena información, esta información es conocimiento recibido del ambiente o de otros agentes. Un conjunto de creencias comunes y revisables

es compartido entre los agentes, este conjunto es la hipótesis actual del sistema. Si un agente recibe información contradictoria entonces intenta modificar la hipótesis actual para mantener la consistencia con su propia información. Dado que la hipótesis debe mantenerse consistente para todos los agentes es necesario mantener interacciones sociales. Durante tales interacciones, el agente que revisa la hipótesis juega el rol de aprendiz, mientras que los otros agentes juegan el rol de críticos. El rol asignado a cada agente varía conforme el proceso de aprendizaje avanza.

El protocolo de revisión en SMILE es secuencial, cuando el agente toma el rol de aprendiz, envía la hipótesis modificada a uno de los críticos y espera una respuesta, durante esta espera alguna otra revisión puede ser llevada a cabo, una vez recibe la respuesta, envía la hipótesis a otros agentes. Este proceso continua hasta que la hipótesis es consistente para todos los agentes. En (Bourgne, Soldano, y Seghrouchni, 2010) el protocolo de Aprendizaje presentado en SMILE es remplazado por una solución más eficiente que involucra interacciones paralelas.

1.2. Planteamiento del problema

Refiriéndose a modelos de clasificación aprendidos de forma distribuida, en minería de datos distribuida existe un trade-off fundamental entre certeza del modelo y costo de inducción del mismo (V. S. Rao, 2009). Si el interés es el costo, lo cual se refiere tanto al costo de cómputo como al de comunicación, se puede procesar cada segmento de los datos de forma local, obteniéndose resultados locales, y más tarde combinar los resultados locales para obtener el resultado final. Esto es fundamentalmente lo que propone Meta-Aprendizaje. Pero si el interés se centra en la certeza de clasificación, se pueden mover todos los datos a un sólo sitio y a partir de ahí obtener un resultado global. Este es un proceso centralizado. Lo anterior produce el resultado más certero, sin embargo, es un proceso costoso.

En el presente, una estrategia de aprendizaje colaborativo para minería de datos distribuida es propuesta, esta estrategia está inspirada en aprendizaje colaborativo de conceptos en SMA. Dicha estrategia es un primer acercamiento que deja abiertas posibilidades para futura exploración. La idea central de la propuesta es tratar de mantener la precisión de una estrategia centralizada, mientras se intentan evadir algunos de sus problemas de costo, sobre todo aquellos relacionados con la transmisión de datos y costo de inducción. Además, dado que la estrategia propuesta es incremental, es idónea para tratar casos donde la producción de datos es continua (online) y se requiere una creación de modelos de aprendizaje también continua.
La estrategia de aprendizaje propuesta parte del supuesto de que es posible crear modelos de clasificación con pocos ejemplos que son equivalentes en cuanto a certeza de clasificación a modelos que se construyen a partir de la totalidad de ejemplos. Para lograr lo anterior, se parte de un modelo base inducido con un mínimo de ejemplos, luego, se considera agregar al modelo sólo aquellos ejemplos cuya clase contradiga la clase predicha

por el modelo base, este proceso es incremental, teniéndose que realizar varias inducciones hasta que no existan ejemplos contradictorios. Mediante este proceso es posible reducir el número de ejemplos transmitidos y necesarios para realizar una inducción y es ideal para casos donde se requiere construir modelos de clasificación de forma continua.

1.3. Hipótesis

La hipótesis de la cual parte este trabajo es la siguiente: un esquema de aprendizaje distribuido basado en la creación de modelos base con un mínimo de ejemplos y enriquecido a partir de ejemplos contradictorios, es una opción viable para mantener la precisión de un esquema de aprendizaje centralizado tradicional, mientras se reduce significativamente el número de ejemplos de entrenamiento utilizados para inducir el modelo.

1.4. Justificación

La estrategia centralizada de minería de datos distribuida resulta ser la que mejor resultados arroja en cuanto a certeza de clasificación, sin embargo, como se señaló anteriormente, es una estrategia costosa que resulta inaplicable en muchas situaciones, especialmente porque requiere de una transmisión elevada de datos, y para bases de datos muy grandes, el costo de inducción es muy alto. El esquema que se propone en el presente, es un primer intento por preservar las bondades de la estrategia centralizada mientras se evitan algunos de sus problemas de costo, sobre todo aquellos relacionados con la transmisión de datos y costes de inducción. Es además en su núcleo una opción más sencilla de implementar y mantener que aquellas utilizada en Meta-Aprendizaje. De encontrarse buenos resultados, nuevas estrategias de minería de datos distribuida, basadas en el esquema propuesto en el presente, pueden ser implementadas e incluso utilizadas en una amplia variedad de dominios.

En el presente trabajo además se propone una nueva aproximación para realizar sistemas de minería de datos distribuida basada en el paradigma de agentes y artefactos. En ésta, los artefactos encapsulan herramientas de minería de datos, heredadas de WEKA (Hall y cols., 2009), que los agentes pueden explotar mientras se encuentran inmersos en el proceso de aprendizaje colaborativo y distribuido. Dicha aproximación resulta de gran valor debido a su naturaleza modular, distribuida y escalable.

1.5. Objetivos

Los objetivos concretos de este trabajo son los siguientes:

- Crear una estrategia de aprendizaje colaborativo distribuido, aplicada a minería de datos distribuida que reduzca significativamente el número de ejemplos de en-

trenamiento necesarios para construir el modelo de clasificación mientras mantiene la certeza de clasificación de una estrategia centralizada tradicional.

- Para dar soporte a la estrategia de aprendizaje propuesta, se plantea crear un sistema que realice minería de datos distribuida basado en el paradigma de agentes y artefactos. Este sistema será una plataforma para crear experimentos que permitan hacer comparaciones entre la estrategia propuesta y una estrategia centralizada tradicional.
- Evaluar la viabilidad de la estrategia de aprendizaje colaborativo propuesta mediante una serie de experimentos que utilizan bases de datos conocidas en el área.

1.6. Alcances y limitaciones

Los experimentos realizados se llevan a cabo en un entorno centralizado, simulandose la distribución de datos. No se presentan pruebas con datos distribuidos geográficamente, por lo tanto no es posible ver grandes contrastes de tiempo en la comunicación de los datos. Dado lo anterior, es de principal interés del presente evaluar la cantidad de ejemplos utilizados para crear el modelo de entrenamiento así como la certeza de clasificación de éstos, se considera el tiempo como medio para comparar la eficiencia del esquema propuesto, sin embargo, debe tomarse en cuenta que estos tiempos no son del todo realistas debido, como ya se mencionó, a que los experimentos se despliegan en un entorno centralizado.

En el presente sólo se considera un clasificador, J48 (Implementación de C4.5 en WEKA). Lo anterior debido a que se juzgó que dada la forma en que éste aumenta su especificidad al encontrarse con nuevos ejemplos, J48 resulta ideal para el esquema propuesto.

1.7. Exposición y método

El documento se encuentra dividido en dos partes. La primera parte (capítulos 2 a 5) presenta el estado del arte en el que se desarrolla el presente trabajo. El capítulo 2 presenta conceptos de agencia, especialmente de agencia BDI, los cuales son relevantes para el presente ya que se implementa un sistema multi-agente. En el capítulo 3 se introduce la programación orientada a agentes, en dicha sección, además de hablar del estado actual del área, se presenta Jason, un lenguaje de programación orientado a agentes, el cual es utilizado en el trabajo presente. Así mismo se introducen conceptos referentes a ambientes endógenos y CArtAgO un exponente de este paradigma que es fundamental en el sistema experimental implementado. El capítulo 4 trata sobre aprendizaje colaborativo de conceptos en SMA, especialmente tratando la propuesta de SMILE. La estrategia de aprendizaje colaborativa propuesta en el presenta está inspirada en dicho trabajo. Finalmente, en el capítulo 5 se introducen conceptos de minería de datos, pasando por árboles

de decisión; minería de datos distribuida, donde se habla también de Meta-Aprendizaje; minería de datos distribuida basada en agentes; y evaluación de modelos de clasificación, presentando validación cruzada y prueba T pareada.

En la segunda parte (capítulos 6-10) se cubre el desarrollo del trabajo. En el capítulo 6 se discute el diseño e implementación de la propuesta, pasando por puntos como son la descripción de la estrategia de aprendizaje colaborativo distribuido, diseño de la plataforma experimental de aprendizaje y un análisis sobre la eficiencia del la estrategia y sistemas propuestos. El capítulo 7 presenta la metodología experimental. En el capítulo 8 se muestran los resultados arrojados a manera de cuadros. En la sección 9 se realiza una discusión acerca de los resultados obtenidos. Finalmente, el capítulo 10 cierra el presente trabajo con una conclusión he indicios de trabajo futuro.

Parte I.

Estado del Arte

2. Agencia

Para entender lo que el término 'agente' y 'sistema multi-agente' significan, es mejor empezar considerando cómo los agentes se relaciones con otros tipos de software. Empezando por considerar programas funcionales, los cuales son posiblemente el tipo de software más simple desde el punto de vista de desarrollo e ingeniería de software. Un programa funcional toma alguna entrada, trata esta entrada, y entonces con base en ésta, produce alguna salida y termina. Los programas funcionales son llamados de esta forma debido a que, matemáticamente, pueden pensarse como funciones. Se cuneta con un rango de técnicas bien establecidas para desarrollara tales programas. Desafortunadamente, muchos programas no pueden ser desarrollados con una estructura operacional simple de entrada - cómputo - salida. En particular, muchos de los sistemas que necesitan ser construidos en la práctica tienen características 'reactivas', en el sentido de que tienen que mantener una interacción continua con su ambiente a largo plazo; estos sistemas no computan simplemente una función y entonces terminan. Los sistemas reactivos son sistemas que no pueden ser descritos adecuadamente desde los puntos de vista relacionales o funcionales. El punto de vista relacional trata a los programas como funciones, desde un estado inicial hasta un estado final. Típicamente, el rol principal de un sistema reactivo es mantener la interacción con su ambiente, y por lo tanto debe ser descrito en términos de su comportamiento continuo. Los sistemas concurrentes son casos especiales de sistemas reactivos. Todos los sistemas concurrentes deben ser estudiados desde el punto de vista de su comportamiento. Esto debido a que cada módulo individual en un sistema concurrente es un subsistema reactivo que interactúa con su propio ambiente que consiste de otros módulos.

Existe una clase de sistema más complejo aun que es un subconjunto de los sistemas reactivos que son llamados agentes (Bordini y cols., 2007). Un agente es un sistema reactivo que exhibe algún grado de autonomía en el sentido de que se le pueden delegar algunas tareas y el sistema por si mismo determina la mejor forma de cumplir dicha tarea. A estos sistemas se le llaman agentes debido a que se piensa en ellos como si estuviesen activos, produciendo acciones con propósito: los agentes son enviados a un ambiente para que cumplan tareas por nosotros, se quiere que éstos se encuentren activamente persiguiendo estas metas, averiguando por si mismos cuál es la mejor forma de cumplir dichas metas, en vez de tener que decirles cómo deben hacer las cosas en una forma de bajo nivel.

Se considera que los agentes son sistemas que están situados en algún ambiente. Esto quiere decir que los agentes deben ser capaces de sensar su ambiente (por medio de sensores), y tienen un repertorio de acciones posibles que pueden ejecutar (por medio de

efectores) para modificar dicho ambiente. La pregunta clave para el agente es cómo ir desde la entra del sensor hasta la salida de la acción: cómo decidir que hacer basándose en la información de los sensores.

El ambiente que el agente ocupa puede ser físico (en el caso de robots que habitan el mundo físico) o de software (en el caso de un agente de software que habita un sistema operativo o red de cómputo). En la mayoría de aplicaciones realistas, los agentes tiene a lo más un control parcial sobre el ambiente. Así, mientras los agentes pueden ejecutar acciones que cambian el ambiente, ellos no pueden en general tener un control completo sobre éste. Esto sucede muchas veces debido a que existen otros agentes en el ambiente, quienes ejercen control sobre su parte del ambiente.

Además de estar situados en un ambiente, se espera que un agente racional tenga las siguientes propiedades (Wooldridge, Jennings, y cols., 1995):

- Autonomía: la autonomía es una característica que puede medirse en un amplio espectro. En un extremo de este espectro, se tienen programas de computadora convencionales como son los procesadores de palabras y hojas de cálculo, que exhiben poca o nula autonomía. Todo lo que sucede en tales aplicaciones sucede debido a la interacción del usuario. Tales programas, no toman ninguna iniciativa en ningún sentido. En el otro extremo del espectro de autonomía estamos los seres humanos, que podemos decidir que hacer y en que creer así como establecer las metas que deseemos por nosotros mismos. En los sistemas basados en agentes autónomos se busca reproducir ciertas características de autonomía, sin embargo, en la actualidad no puede esperarse una autonomía total por lo que se esperan programas de computadora que se encuentren de alguna forma en el medio de los dos extremos antes mencionados. En otras palabras, se quiere tener la posibilidad de delegar metas a los agentes, los cuales deciden cual es la mejor forma de actuar para conseguir esas metas. Así, la habilidad del agente para construir metas está directamente ligada a las metas que se le delegan. Mas específicamente, la forma en que los agentes actuarán para cumplir sus metas está ligada a los planes que se le proveen al agente, que definen las formas en que un agente puede actuar para cumplir metas y submetas.

  En su forma más simple, la autonomía significa ser capaz de operar independientemente para lograr metas que se le delegan al agente. Así, un agente autónomo realiza decisiones independientes acerca de cómo lograr sus metas delegadas, sus decisiones y acciones están bajo su propio control y no el de otros agentes.

- Proactividad: esta característica significa ser capaz de exhibir comportamiento dirigido al cumplimiento de metas. Si a un agente se le a delegado una meta en particular, entonces se espera que el agente intente lograr dicha meta. La proactividad excluye a los agentes pasivos, que nunca intentan hacer nada. Así, no se piensa en objetos, en el sentido de Java, como agentes: tales objetos son esencial-

mente pasivos hasta que algo invoca un método en ellos, osea hasta que algo o alguien les dice que hacer. Lo mismo aplica para otras entidades de cómputo como los servicios web.

- Reactividad: ser reactivo significa ser responsivo a los cambios en el ambiente. En la vida diaria, un plan raramente funciona sin problemas. Un plan es frecuentemente frustrado, accidental o deliberadamente. Cuando alguien se da cuenta de que uno de sus planes va mal, responde escogiendo cursos de acción alternativos. Algunas de esas respuestas están al nivel de los reflejos, mientras que otras requieren de mayor deliberación. Desarrollar un sistema que simplemente responde a los estímulos del ambiente de una forma basada en reflejos directos no es difícil. Puede implementarse como una tabla que mapea un estado del sistema a una acción. Así mismo, implementar un sistema puramente dirigido al cumplimiento de metas tampoco es difícil, esto es lo que los programas de cómputo convencionales son. Sin embargo, implementar un sistema que logre un balance efectivo entre comportamiento reactivo y dirigido a metas es complicado.
- Habilidad social: todos los días, millones de computadoras alrededor del mundo intercambian información de forma rutinaria. En este sentido, construir sistemas de cómputo que tengan algún tipo de habilidad social no es complicado. Sin embargo, la habilidad de intercambiar bytes no es habilidad social en el sentido que se desea para un agente racional. Lo que se quiere es la habilidad para que los agentes puedan cooperar y coordinar actividades con otros agentes, de tal forma que puedan cumplir nuestras metas. Para lograr este tipo de habilidad social, es útil tener agentes que puedan comunicarse no sólo en términos de intercambio de bytes o invocando métodos, sino que puedan comunicarse al nivel de conocimiento. Esto es, se quieren agentes capaces de comunicar a oitros agentes sus creencias, metas y planes.

En la práctica es común encontrar ambientes donde varios agentes existen. Este es el tipo de ambientes de interés de los sistemas multi-agente. Cada agente del sistema tiene una esfera de influencia, esta esfera determina la fracción del ambiente que el agente puede controlar, o controlar parcialmente. Puede darse el caso que las esferas de influencia se superpongan, esto ocurre cuando más de una agente tiene control sobre la misma parte del ambiente. Este tipo de problemas le complica la vida a los agentes ya que para conseguir un resultado deseado en el ambiente, los agentes deben considerar el como los otros agentes actuaran. Los agentes pueden organizarse por medio de relaciones entre ellos (por ejemplo, un agente puede obedecer a otro agente). Los agentes pueden tener conocimiento de otros agentes, pudiéndose dar el caso de que sólo conozcan a algunos de los agentes del sistema.

2.1. Agentes BDI

El modelo creencia-deseo-intención (BDI por sus siglas en ingles, belief-desire-intention) está basado en el comportamiento humano y fue desarrollado por filósofos. La arquitectu-

ra BDI tiene sus orígenes en el proyecto de Agencia Racional del instituto de investigación de la universidad de Stanford de mediados de 1980. Los orígenes del modelo descansan en la teoría de razonamiento humano práctico desarrollado por el filosofo Michael Bratman (Bratman, 1987), el cual se enfoca particularmente en el rol que juegan las intenciones en el razonamiento práctico. El framework conceptual del modelo BDI está descrito en (Bratman, Israel, y Pollack, 1988), en donde también se describe una arquitectura de agente especifica llamada IRMA.

La distinción entre creencias, deseos e intenciones es la siguiente:

- Las creencias son información que el agente tiene acerca del mundo. Esta información puede ser imprecisa.
- Los deseos son posibles estados que al agente le gustaría conseguir. Sin embargo, tener un deseo no implica que el agente actuará para cumplirlo: es una influencia potencial de las acciones del agente. Es razonable para un agente racional tener deseos que son mutuamente incompatibles entre ellos. Normalmente se habla de los deseos como opciones que el agente tiene.
- Las intenciones son estados del mundo para los que el agente se encuentra trabajando de tal forma que pueda conseguirlos. Las intenciones pueden ser metas delegadas al agente, o pueden ser el resultado de la consideración de opciones: puede pensarse que el agente realiza una evaluación de sus opciones y escoge entre ellas. Las opciones seleccionadas de esta forma se convierte en intenciones. El agente comienza con una meta delegada, y entonces considera sus posibles opciones compatibles con la meta delegada.

De acuerdo con Dennett (Dennett, 1989), en la literatura de filosofía, el término sistema intencional es utilizado para referirse a sistemas cuyo comportamiento puede ser predecido y explicado en términos de actitudes tales como las creencias, deseos e intenciones. La Justificación racional de esta aproximación es que , en la vida diaria, utilizamos un tipo de psicología popular para explicar y predecir el comportamiento de otras personas. Por ejemplo, puede decirse que Miguel intenta escribir un artículo para explicar su comportamiento. Una vez se dice esto, se espera encontrar a Miguel diseñando un plan y realizando acciones que lo lleven a escribir dicho artículo; se puede esperar que pase mucho tiempo en su computadora; no sería sorprendente que se encontrase de mal humor; pero sería sorprendente encontrarlo en una fiesta nocturna.

Esta postura intencional, donde el comportamiento de un sistema complejo es entendido por la atribución de actitudes tales como creer y desear, es simplemente una herramienta de abstracción. Es una forma conveniente de hablar acerca de sistemas complejos, que permite predecir y explicar su comportamiento sin la necesidad de entender como funciona en realidad. Actualmente las ciencias computacionales se encuentran buscando buenos mecanismos de abstracción, dado que éstos permiten a los desarrolladores de sistemas manejar la complejidad con mayor facilidad. Así, la postura intencional

se postula como herramienta de abstracción en computación para explicar, entender y programar sistemas computacionales complejos.

2.1.1. Razonamiento práctico

Las estructuras de datos clave para los agentes son las creencias, deseos e intenciones. ¿Cómo puede un agente con creencias, deseos e intenciones realizar la transición de estos elementos a acciones?. El modelo particular de toma de decisiones del modelo BDI es conocido como razonamiento práctico. El razonamiento práctico es razonamiento dirigido a las acciones, es el proceso de descifrar lo que debe de hacerse. En razonamiento práctico deben considerarse situaciones de conflicto donde se tienen opciones compitiendo, en estos conflictos las consideraciones relevantes son provistas por lo que el agente desea y por lo que el agente cree.

El razonamiento práctico humano parece consistir en dos actividades distintas: deliberación (determinar el estado de cosas que se desea conseguir, es decir, las intenciones; y razonamiento medios-fines (decidir como actuar para conseguir las metas de las intenciones).

Deliberación e intenciones
El proceso de deliberación resulta en la adopción de intenciones por parte del agente. Las intenciones son actitudes pro-activas: tienden a llevar a la acción. Si se tiene la intención de escribir un libre, entonces es de esperarse que se realice algún intento para realizar tal intención. Bratman denota que las intenciones juegan un rol más fuerte en la influencia de acciones que otras actitudes pro-activas, tales como los deseos.

La segunda propiedad principal de las intenciones es que persisten. Si se adopta una intención, es de esperarse que se persista con esta intención en un intento de conseguirla. De desechar la intención inmediatamente sin haberle dedicado ningún tipo de esfuerzo, sería fácil inclinarse a decir que en realidad nunca se tuvo tal intención. Por otro lado, tampoco es deseable persistir demasiado tiempo en una intención, si se vuelve claro que nunca se podrá lograr la meta, o si las razones para perseguir la meta ya no son válidas, entonces es racional abandonar la intención.

La tercera propiedad principal de las intenciones es que, una vez que han sido adoptadas, el mismo hecho de tener dicha intención restringirá el razonamiento práctico futuro del agente. Mientras se mantiene alguna intención en particular, no se evaluarán opciones que son inconsistentes con dicha intención. Así, las intenciones provén un 'filtro de admisibilidad' ya que restringen el espacio de posibles intenciones que el agente necesita considerar.

Finalmente, las intenciones están relacionadas fuertemente con creencias acerca del futuro. En particular, intentar algo implica que se cree que en un principio ese algo es

posible, y que, en circunstancias normales, se tendrá éxito.

Razonamiento Medios-Fines
El razonamiento medios-fines es el proceso de decidir cómo conseguir un fin (es decir, una intención escogida) utilizando los medios disponibles (es decir, las acciones que se pueden ejecutar en el ambiente). El razonamiento medios-fines es quizás mejor conocido en la comunidad de IA como planificación (Ghallab, Nau, y Traverso, 2004). Un planificador es un sistema que recibe como entrada representaciones de:

- Una meta, o intención: algo que el agente quiere conseguir
- Las creencias actuales del agente acerca del estado del ambiente
- Las acciones disponibles para el agente

Como salida, el algoritmo de planificación genera un plan. Un plan es un curso de acción. Si el algoritmo de planificación realiza su tarea correctamente, entonces si el agente ejecuta su plan desde un estado donde el mundo es descrito por las creencias actuales del agente, entonces una vez que el plan a sido ejecutado por completo, la meta del agente habrá sido conseguida.

El enfoque original en IA fue la producción de planes a partir de 'primeros principios' (Tate, Hendler, y Allen, 1994). Por éstos se entiende el ensamble de cursos de acción completos (un programa) en el cual los componentes atómicos son las acciones disponibles para el agente. El mayor problema de esta aproximación es que resulta computacionalmente muy costosa: incluso con restricciones poco realistas, la planificación de este tipo es PSPACE-completa, y por lo tanto más compleja que un problema NP-duro tales como el problema del vendedor viajero. Con menos restricciones y asunciones más realistas, la planificación es mucho más compleja. A pesar de que avances significativos se han realizado enfocados al desarrollo de algoritmos eficientes para este tipo de planificación, su complejidad inherente hace dudar a los investigadores si dichos algoritmos de planificación deberían ser utilizados en ambientes de toma de decisiones donde los agentes deben planear y actuar en tiempo real. En sistemas multi-agente el enfoque se simplifica de una forma que ha demostrado ser bastante poderosa en la práctica: la idea es que el programador desarrolle una colección de planes parciales para los agentes en tiempo de diseño, y la tarea del agente será ensamblar dichos planes en tiempo de ejecución para lidiar con la meta en la actual el agente se encuentra actualmente trabajando. Esta aproximación puede parecer una variación simple de planificación de primeros principios (en lugar de ensamblar acciones se ensamblan planes); y parece además carecer de flexibilidad. Sin embargo, este modelo a resultado bueno en la práctica.

3. Programación Orientada a Agentes

La idea de programar sistemas computacionales en términos de nociones 'mentales' tales como creencias, deseos e intenciones es un componente clave del modelo BDI. El concepto fue articulado por Shoham, en su propuesta de programación orientada a agentes (Shoham, 1993). Existen argumentos a favor de la programación orientada a agentes:

- Ofrece una forma familiar, no técnica de hablar acerca de sistemas complejos. No es necesario tener un entrenamiento formal para entender conceptos de mentalidad: es parte de nuestra habilidad lingüística diaria
- La programación orientada agentes puede ser encapsulada en un tipo de programación 'post-declarativa'. En programación procedural, decir lo que un sistema debe hacer implica indicar de forma precisa cómo se debe hacer, escribiendo un algoritmo detallado. La programación procedural es difícil porque a las personas les cuesta trabajo pensar en términos de los detalles requeridos. En programación declarativa (estilo Prolog), el objetivo es reducir el énfasis en los aspectos de control: se especifica una meta que se desea el sistema cumpla, y se deja el trabajo de lo que se debe hacer para cumplir la meta al mecanismo de control interno. Sin embargo, para ser capaz de escribir programas eficientes o largos en un lenguaje como Prolog, es necesario para el programador conocer y entender a detalle el funcionamiento del mecanismo de control interno. Esto entra en conflicto con uno de los objetivos principales de la programación declarativa: quitarle la necesidad al usuario de tratar con cuestiones de control. En programación orientada a agentes, la idea es que, tal como en programación declarativa, se establezcan metas, y dejar al mecanismo de control interno el trabajo de lo que se debe hacer para cumplir la meta. En este caso, el mecanismo de control implementa algún modelo de agencia racional. Idealmente, los programadores del paradigma de programación orientada a gentes, no deberían preocuparse por el cómo los agentes cumplen sus metas (aunque en la práctica este no es siempre el caso).

La investigación en sistemas multi-agente (MAS) ha llevado al desarrollo de lenguajes pacticos y herramientas de programación que son apropiadas para implementar sistemas multi-agente. Llegar a cimentar este paradigma de programación se ha convertido en uno de los tópicos más importantes de investigación en esta área, particularmente porque este es un requerimiento esencial para una eventual transferencia tecnológica (Bordini y cols., 2006).

Al revisar la literatura de sistemas multi-agente se pueden encontrar una gran cantidad de propuestas de lenguajes orientados a agentes, desde lenguajes puramente declarativos, hasta puramente imperativos, existiendo además híbridos. Algunos de estos lenguajes son diseñados desde cero, tratando de plasmar alguna teoría de agencia, mientras que otros extienden a lenguajes existentes tratando de adaptarlos a las particularidades de este paradigma.

Utilizar estos lenguajes, en vez de lenguajes más convencionales, es conveniente cuando el problema es modelado como un sistema multi-agente y entendido en términos de conceptos cognitivos y sociales tales como creencias, metas, planes, roles y normas. La mayoría de lenguajes de programación orientados a agentes están contenidos en una plataforma que implementa su semántica. En cualquier caso, existen frameworks de agente que no se encuentran fuertemente acoplados con algún lenguaje de programación especifico (véase CArtAgO por ejemplo). En lugar de ello, estos frameworks buscan proveer técnicas generales para aspectos relevantes tales como comunicación entre agentes y coordinación. La mayoría de lenguajes de programación orientados a agentes maduros están acompañados de un ambiente integrado de desarrollo (IDE por sus siglas en ingles), que tienen por objetivo mejorar la productividad de los programadores al automatizar tareas de codificación. Típicamente, estos IDEs proveen funcionalidades tales como manejo del proyecto, creación y edición de archivos fuente, refactorización, compilación y ejecución, pruebas y depuración.

A pesar de la gran cantidad de lenguajes, frameworks, ambientes de desarrollo, y plataformas recientemente propuestas, implementar un sistema multi-agente es aun una tarea complicada. Para resolver el problema del manejo de la complejidad inherente de un sistema multi-agente y para ayudar a estructurar su desarrollo, la comunidad de investigación ha producido varias metodologías (Bernon, Cossentino, y Pavón, 2005). De cualquier forma, incluso si los desarrolladores de sistemas multi-agente siguen tales metodologías durante la etapa de diseño, éstos encuentran grandes dificultades en la etapa de implementación, esto ocurre parcialmente debido a la falta de madurez tanto de la metodología como de las herramientas de programación. Entre otras cosas,tales dificultades pueden deberse a la falta de herramientas de depuración apropiadas; a la falta de habilidades necesarias para el mapeo entre conceptos de análisis/diseño y constructores del lenguaje de programación; la falta de competencia al lidiar con características especificas de diversas plataformas de agente; y también a la falta de entendimiento de los fundamentos así como de las características prácticas de la programación orientada a agentes.

A pesar de que la mayoría de lenguajes y herramientas desarrolladas hasta el momento no han sido aun probadas a gran escala, mucho progreso ha sido logrado en los últimos años. Actualmente se vive un gran momento para consolidar esta área y guiar su desarrollo futuro.

3.1. JASON

Jason pertenece a los lenguajes híbridos de programación orientados a agentes, exhibiendo características de lenguajes declarativos e imperativos. Jason es un interprete, implementado por R. Bordini y J. Hübner, es una versión extendida de Agebt Speak(L), un lenguaje de programación orientado a agentes introducido por A. Rao en (A. S. Rao, 1996). El lenguaje está influenciado por el trabajo en arquitecturas de Creencias-Deseaos-Intenciones (BDI) y lógicas BDI (A. S. Rao, Georgeff, y cols., 1995). Jason es de fuente abierta (Open Source) bajo la licencia LGPL y se encuentra disponible en http://jason.sourceforge.net.

Algunas de las opciones disponibles en Jason son:

- Comunicación inter-agente basada en actos de habla y anotaciones en las creencias que especifican el origen de la información
- Anotaciones en las etiquetas de los planes, las cuales pueden ser utilizadas por funciones de selección personalizadas
- Funciones de selección personalizadas en Java, funciones de confianza, arquitecturas de agente personalizadas
- Extensibilidad directa mediante "acciones internas" definidas por el usuario
- Una noción clara de ambiente multi-agente, el cual es implementado en Java. Este ambiente puede ser utilizado para simular un ambiente real.
- Jason cuenta con un IDE así como con un plug-in para el IDE Eclipse. Con éstos es posible editar archivos tanto Java como AgentSpeak para los agentes individuales. A través de IDE es también posible ejecutar y controlar la ejecución del sistema multi-agente y distribuir agentes sobre la red de una forma muy simple. El IDE brinda una opción conocida como el "Mind Inspector" que permite al usuario inspeccionar los estados internos del agente cuando el sistema se encuentra corriendo en modo depuración.

3.1.1. Arquitectura de Jason

Es conveniente realizar una distinción entre programa de agente y arquitectura de agente. La arquitectura de agente es el framework de agente sobre el cual un programa de agente corre. El programador escribe programa que dirigirá el comportamiento del agente, pero mucho de lo que el agente efectivamente realiza es determinado por la propia arquitectura, sin la necesidad de que el programador se preocupe de ello. Por ejemplo, siendo Jason una extensión de AgentSpeak, el cual está basado en la arquitectura BDI, entonces uno de los componentes de la arquitectura de agente es la base de conocimientos, este componente se encuentra actualizándose constantemente sin la necesidad de

que el programador se preocupe por éste.

Los agentes BDI prácticos son sistemas de planificación reactiva (Bordini y cols., 2007). Este tipo de sistemas está diseñado para estar permanentemente en ejecución, reaccionando a alguna forma de evento. La forma en que los sistemas de planificación reactiva reaccionan a eventos es mediante la ejecución de planes; los planes a su vez son cursos de acciones que el agente se compromete a ejecutar para poder responder a los eventos. Las acciones cambian el ambiente del agente , de tal forma que puede esperarse que las metas de losa agentes sean logradas. Un agente se encuentra constantemente percibiendo su ambiente, razonando como actuar de tal forma que sea capaz de cumplir sus metas y entonces actuando para cambiar su ambiente. La parte de razonamiento práctico del ciclo de comportamiento de un agente es llevado a cabo de acuerdo a los planes que el agente tiene definidos en su biblioteca de planes. Esta biblioteca se es definida por el programador y es la parte principal del programa de agente.

Jason cuenta con varios construcciones los cuales pueden ser divididos en tres categorías principales:

- Creencias
- Metas
- Planes

A continuación se presenta una descripción general de estas categorías.

Creencias
Siendo AgentSpeak un lenguaje basado en lógica de primer orden, la representación de información se lleva a cabo en una forma simbólica utilizando predicados, tales como:
$alto(juan)$.
que expresa la propiedad ser alto de un objeto o persona, en este caso 'Juan'. Para expresar una relación entre dos objetos puede utilizarse un predicado como:
$gustar(juan, musica)$.
que indica que ha Juan le gusta la música. Una *literal* es cualquier predicado o su negación. Un agente cuenta con una base de creencias que a su vez está conformada de literales que representan información ya sea sobre el medio ambiente, otros agentes o si mismo.

Tal como en Prolog, cualquier símbolo (secuencia de caracteres) que empieza con minúsculas es llamado *átomo*, los cuales son utilizados para representar objetos o individuos particulares. Los átomos son equivalentes a las constantes de la lógica de primer orden. A un símbolo que comienza con mayúsculas se le denomina *variable*, inicialmente las variables están libres o no instanciadas y una vez que son instanciadas o ligadas a un valor en particular, éstas mantienen su valor a través de su ámbito, en el caso de AgentSpeak el ámbito es el plan en el que aparecen. Las variables se ligan con un valor a través

de la operación de *unificación*; una fórmula es llamada *ground* cuando no tiene variables sin instanciar. La palabra término es utilizada para referirse a constantes, variables y estructuras. Una estructura es utilizada para representar datos complejos, las estructuras comienzan con un átomo denominado *functor* y es seguido por un número de términos separados por comas y delimitados por paréntesis llamados *argumentos*. También existe una estructura especial conocida como *lista*, la definición y uso de esta estructura es similar a la de Prolog. Además de átomos, los número y cadenas se clasifican como constantes. La figura 6.3 muestra la jerarquía de tipos de términos de AgentSpeak que pueden aparecer en una literal de Jason. Para un tratamiento más amplio de lógica y programación lógica ver (Bratko, 2001), (Clocksin y Mellish, 2003), (Lloyd, 1984).

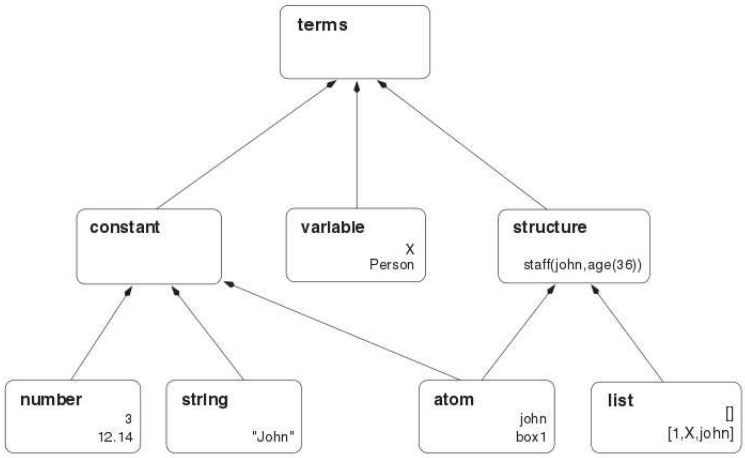

Figura 3.1.: Tipos de términos AgentSpeak en Jason

Una característica interesante que agrega Jason con respecto a la programación lógica tradicional son las *anotaciones*. Las anotaciones son términos complejos que proveen detalles que están asociados fuertemente con una creencia. Las anotaciones están delimitadas por corchetes siguiendo inmediatamente a una literal, por ejemplo:
$ocupado(juan)[expira(verano)]$
lo cual puede significarse que un agente tiene la creencia de que Juan se encuentra ocupado pero que esa creencia expira en verano. Con este simple ejemplo puede pensarse que las anotaciones no incrementan el poder expresivo del lenguaje, bien pudo crearse un predicado de la forma:
$ocupado\_hasta(juan, verano)$
lo que significa lo mismo. Sin embargo, existen dos razones para preferir el uso de anotaciones. La primera es simplemente la elegancia de la notación, lo cual puede ser de gran valor para el programador. La segunda razón es que esta representación también facilita el manejo de la base de creencias. Por ejemplo, la anotación expira, puede influenciar el tiempo de duración que la creencia $ocupado(juan)$ estará en la base de creencias. Así mismo, existen anotaciones especiales para el interprete, por ejemplo la anotación *source*

que sirve para indicar la fuente a partir de la cual una creencia se generó. Existen tres tipos diferentes de fuentes de información para los agentes en un sistema multi-agente:

- Información perceptual: la creencia se genera a partir de la observación del ambiente
- Comunicación: un agente puede comunicarle una creencia a otro agente por medio de actos de habla
- Notas mentales: son creencias que el agente auto genera para recordar cosas del pasado, o cosas que han prometido realizar. Son útiles en procesos que pueden ser largos.

Otro tipo de creencias manejado por Jason son las reglas las cuales son similares a las reglas de Prolog. Las reglas permiten concluir nuevas cosas a partir de cosas que se conocen. Incluir reglas en la base de creencias de un agente puede simplificar algunas tareas, por ejemplo al utilizar alguna condición en un plan. A diferencia de Prolog, en Jason la notación de reglas es extendida para soportar anotaciones, de tal forma que es posible realizar evaluaciones sobre variables que aparecen en una anotación, por ejemplo:

$$guardar\_pertenencia(P) :- pertenencia(P)[source(S)]\&(S == self | S == juan).$$

La anterior regla indica que el agente puede guardar una pertenencia si ésta es propia o si es de Juan.

A pesar de que los programadores de AgentSpeak pueden manipular la base de creencias de la forma que crean conveniente, en AgentSpeak, sólo es necesario proveer las creencias y reglas (de haberlas) iniciales. Estas creencias iniciales son cargadas por el interprete en el momento de la creación del agente, antes incluso de que el agente sea capaz de percibir su ambiente. Las creencias iniciales son añadidas por defecto como notas mentales.

Metas

En programación orientada a agentes la noción de meta es fundamental. Mientras que las creencias, especialmente aquellas percibidas del ambiente, expresan propiedades que se cree son verdaderas para el mundo en el cual el agente se encuentra situado, las metas expresan las propiedades de los estados del mundo que el agente desea volver verdaderas. Cuando se representa una meta m en un programa de agente, esto significa que el agente está comprometido a actuar de tal forma que sea capaz de cambiar el mundo a un estado en el cual crea, al sensar el ambiente, que m es en efecto verdadero. Este uso particular de las mentas es referido en la literatura de programación de agentes como *meta declarativa*

En AgentSpeak existen dos tipos de metas: *metas de logro* (achievement goals) y *metas de prueba*(test goals). Las metas de logro se distinguen por el operador '!'. Así, por ejemplo, puede decirse !poseer(casa) para establecer que el agente tiene la meta de lograr un

cierto estado del mundo en el que sea capaz de creer que posee una casa, lo que implica que probablemente en la actualidad el agente no cree que posea una casa. El hecho de que un agente adopte una nueva meta puede implicar la ejecución de planes por parte de este, los cuales son cursos de acciones que el agente espera traigan como consecuencia el cambio en el estado del mundo deseado, osea que cumplan la meta deseada. Las metas de prueba son utilizadas normalmente para devolver información de la base de creencias. Así por ejemplo, cuando se escribe ?$balance\_bancario(BB)$, se hace típicamente porque se desea que la variable lógica BB sea instanciada con el valor que actualmente se cree le corresponde al balance bancario. En ciertas circunstancias las metas de prueba también pueden llevar a la ejecución de planes.

Existe una distinción importante entre la noción de meta de Prolog y la noción de meta en AgentSpeak. Una meta o clausula de meta en Prolog es una conjunción de literales que se desea el interprete de Prolog verifique para saber si pueden ser concluidas a partir del conocimiento base representado en el programa; el interprete de Prolog está esencialmente probando si la clausula es una consecuencia lógica del programa.

Planes

Un plan de AgentSpeak tiene tres partes distintas: el *evento disparador* (triggering event), el *contexto*(context), y el *cuerpo*(body). Juntos el evento disparador y el contexto son llamados la *cabeza* del plan. Las tres partes de un plan están separadas sintácticamente por ':' y '¡-' como sigue:

$evento\_disparador : contexto < -cuerpo.$

A continuación se explica brevemente cada uno de los elementos mencionados:

- Evento disparador: Existen dos aspectos importantes del comportamiento de un agente: reactividad y proactividad. Los agentes tienen metas que intentan cumplir a largo plazo, esto determina el comportamiento proactivo de éste. Sin embargo, mientas el agente actúa de tal forma que sea capaz de cumplir sus metas, el agente necesita prestar atención a los cambios que el ambiente sufre, ya que esos cambios pueden determinar si el agente será capaz de lograr sus metas. Los cambios n el ambiente pueden significar que existen nuevas oportunidades para el agente, estas oportunidades pueden llevarlo a reconsiderar cómo llevar a cabo su trabajo, adoptando nuevas metas a corto plazo o descartando otras.

  Existen dos tipos de cambios en la actitud mental de un agente que son importantes en un programa de agente: cambios en las creencias y cambios es las metas del agente. Cambios en ambos tipos de actitudes crean, dentro de la arquitectura del agente, los eventos sobre los cuales los agentes actuarán. Así mismo, tales cambios pueden ser de dos tipos: *adición* y *eliminación*.

La parte del evento disparador en un plan existe precisamente para decirle al agente, para cada plan de su biblioteca, cuáles son los eventos específicos para los cuales el plan es usado. Si un evento que acaba de ocurrir concuerda con el evento disparador del plan, ese plan es candidato para ser ejecutado. Si el evento disparador de un plan concuerda con un evento en particular, se dice que el plan es *relevante* para ese evento.

- Contexto: Así como el evento disparador, el contexto también juega un papel importante en la reactividad del sistema de planes. Los agentes tienen metas y utilizan los planes para cumplir dichas metas, pero también necesitan estar atentos a cambios en el ambiente. Es difícil lidiar con ambientes dinámicos en primer lugar debido a que los cambios en el ambiente pueden significar que es necesario actuar más, pero también debido a que, a medida que el ambiente cambia, los planes que tienen más probabilidades de tener éxito en una meta en particular también cambian.

  Por esta razón por la cual los sistemas de planificación reactiva posponen el comprometerse a un curso de acción (plan), de tal forma que pueda cumplirse una meta lo mas tarde posible; esto es, la elección de plan escogido para una de las muchas metas que el agente tiene se lleva a cabo cuando el agente esta por actuar sobre dicha meta. Típicamente un agente tiene varios planes para cumplir la misma meta.

  El contexto del plan es utilizado para verificar la situación actual para determinar si un plan en particular, de entre varios alternativos, tiene probabilidades de ser exitoso en el manejo del evento, dada la última información que el agente tiene del ambiente. De esta forma, un plan es el elegido para ser ejecutado si su contexto es una *consecuencia lógica* de las creencias del agente. Un plan cuyo contexto evalúa verdadero dado el estado actual de la base de creencias del agente se dice que es *aplicable* en ese momento, y es candidato para ser ejecutado.

- Cuerpo: es simplemente una secuencia de fórmulas que determinan un curso de acción, uno que con optimismo podrá cumplir la meta. Sin embargo, cada fórmula en el cuerpo del plan no es necesariamente una acción directa a ser ejecutada por los efectores del agente (para de esta forma cambiar el ambiente). Además de acciones, en el cuerpo de un plan pueden aparecer sub-metas que deben ser cumplidas para que el plan actual tenga éxito. Así mismo, en el cuerpo de un plan pueden aparecer estructuras de control y expresiones propias del lenguaje.

Los planes así mismo pueden estar etiquetados. Esta etiqueta sirve para identificar el plan. Así mismo a la etiqueta se le pueden agregar anotaciones para hacer uso de directivas especiales para el interprete (como ejecución atómica del plan) o para que el programador ponga sus propias anotaciones y haga uso de ellas en alguna función personalizada. La etiqueta de un plan debe aparecer antes del evento disparador y debe

iniciar con '@'. La figura 3.1.1 muestra un ejemplo de plan completo para el problema clásico del mundo de los bloques.

```
@put1
+!put(X,Y)
    : clear(X) & clear(Y)
    <- move(X,Y).
```

Figura 3.2.: Ejemplo de plan para el problema clásico del mundo de los bloques

3.1.2. Ciclo de razonamiento en Jason

Un agente opera por medio de su ciclo de razonamiento, el cual en el caso de Jason puede ser dividido en 10 pasos principales. La figura 5.3 muestra la arquitectura de un agente Jason así como las funciones que son ejecutadas durante el ciclo de razonamiento. En esta figura, los rectángulos representan los principales componentes arquitectónicos que determinan es estado del agente, esto es, la base de creencias, el conjunto de eventos, la biblioteca de planes y el conjunto de intenciones. Los recuadros redondeados, diamantes y círculos representan funciones utilizadas en el ciclo de razonamiento. Los recuadros redondeados y los diamantes representan funciones que pueden ser modificadas por los programadores, mientras los círculos son partes esenciales del interprete que no pueden ser modificadas. La diferencia entre los recuadros redondeados y los diamantes es que éstos últimos denotan funciones de selección. Las funciones de selección toman una lista de elementos y la función selecciona uno de ellos.

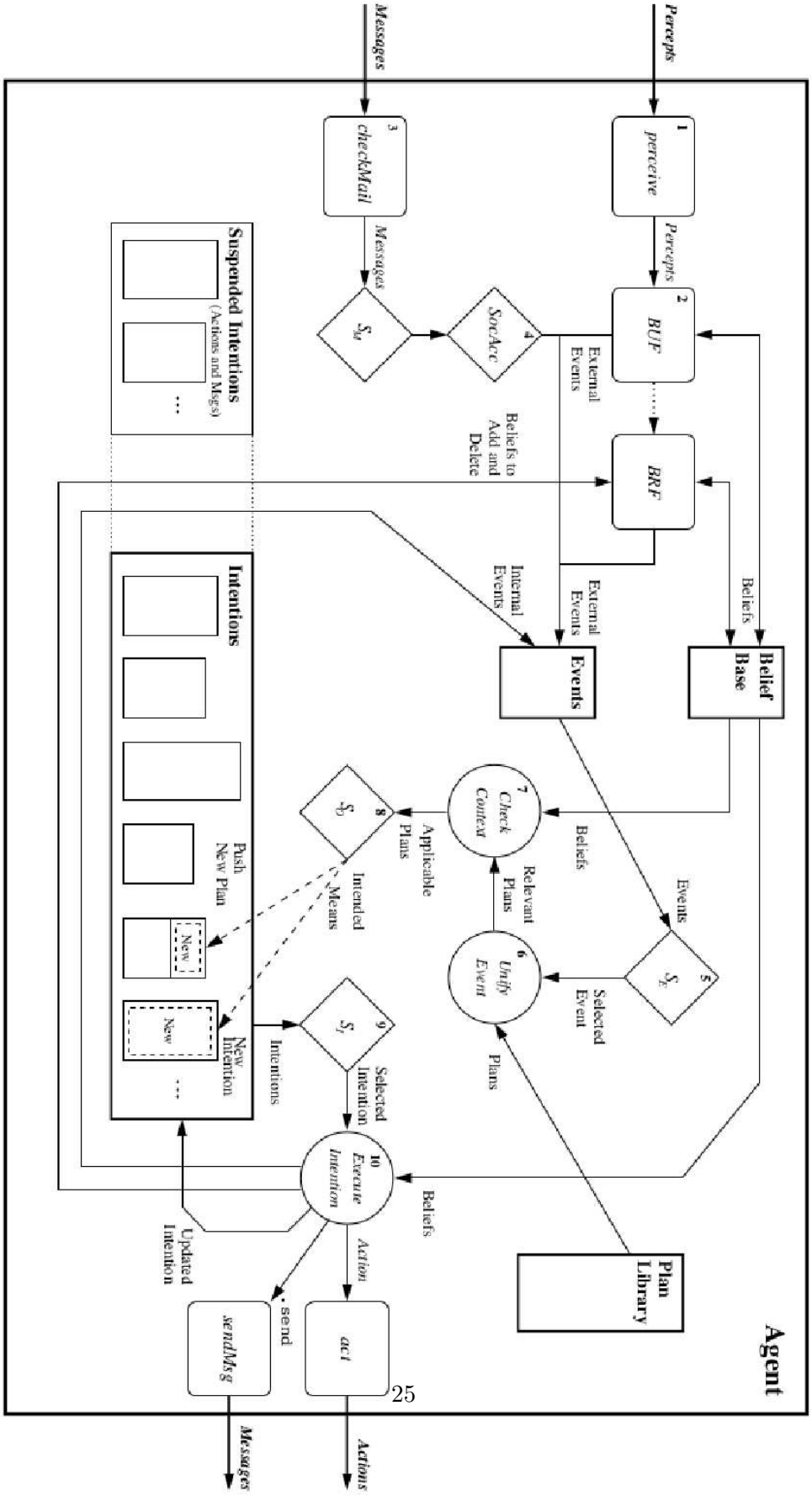

Figura 3.3.: Ciclo de razonamiento de un agente Jason

A continuación se explica cada uno de los pasos involucrados en el ciclo de razonamiento:

- Paso 1 - Percibir el ambiente: la primera tarea que un agente realiza dentro de su ciclo de razonamiento es sensar el ambiente para actualizar sus creencias acerca del estado el ambiente. La arquitectura del agente debe tener un componente que es capaz de percibir el ambiente, de una forma simbólica como listas de literales. Cada literal es una percepción, la cual es una representación simbólica de una propiedad particular del estado actual del ambiente.

  El método de 'perceive' es usado para implementar el proceso de obtención de tales percepciones. En la Implementación por defecto de Jason, el método devuelve la lista de literales del ambiente simulado implementado en Java. En una aplicación donde, por ejemplo, el agente tenga acceso a datos de un sensor situado en el mundo real, el método 'perceive' necesitará tener una interfaz con el sensor.

- Paso 2 - Actualizar la base de creencias: una vez la lista de percepciones ha sido obtenida, la base de creencias necesita ser actualizada para reflejar los cambios percibidos en el ambiente. Lo anterior se logra mediante una función de actualización de la base de creencias, el método que implementa esta función se llama 'buf', el cual puede ser personalizado. El método buf por defecto asume que todo lo que es actualmente percibido en el ambiente será incluido en la lista de percepciones obtenida en el paso previo; esto es, cuando el agente sensa el ambiente, percibe todo lo que es actualmente perceptible por él. Consecuentemente, todo lo que el método buf hace es actualizar la base de creencias es agregar las literales percibidas que no se encuentran en la base de creencias, o borrar aquellas literales que no aparecen en las percepciones pero si en la base de creencias. Cada cambio en la base de creencias genera un evento

- Paso 3 - Recibir comunicación de otros agentes: otra fuente de información importante para el agente en un sistema multi-agente son los otros agentes del mismo sistema. En este punto del ciclo de razonamiento, el interprete verifica los mensajes que pueden haber sido enviados al 'buzón' del agente. Esta operación es llevada a cabo mediante el método 'checkMail', el cual puede ser personalizado. Este método simplemente toma los mensajes almacenados y los vuelve disponibles al nivel del interprete AgentSpeak.

  En un ciclo de razonamiento, sólo un mensaje es procesado por el interprete AgentSpeak. En muchos escenarios, los agentes desearán dar prioridad a ciertos mensajes, y es por esto que se necesita una función de selección de mensajes especifica para el agente. Esta función de selección puede ser personalizada por el programador, por defecto la función regresa los mensajes ene l orden en que fueron recibidos, comenzando por el primero.

- Paso 4 - Seleccionar mensajes "socialmente aceptables": Antes de que los mensajes sean procesados, pasan por un proceso de selección para determinar si pueden o no ser aceptados por el agente. La función del interprete que lleva a cabo esta operación es llamada función de aceptación social y es implementada por un método llamado 'SocAcc'. Este método necesita normalmente ser modificado por el programados, posiblemente para cada tipo de agente. Una implementación en particular del método determina cuales agentes pueden proveer información y saber de cómo realizar una tarea, así como delegación de metas, al agente usando esa implementación del método SocAcc. La Implementación por defecto simplemente acepta mensajes de todos los agentes.

- Paso 5 - Seleccionar un evento: los agentes BDI prácticos operan al manejar eventos de forma continua, que representan cambios percibidos ene l ambiente o cambios en las propias metas del agente. En cada ciclo de razonamiento, sólo un evento pendiente es tratado. Puede haber varios eventos pendientes, por ejemplo debido a que varios aspectos del ambiente cambiaron recientemente y el agente no ha realizado suficientes ciclos de razonamiento para manejar todos los cambios. Por el motivo anterior, es necesario seleccionar el evento que será manejado en un ciclo de razonamiento en particular. Esto se lleva a cabo mediante una función de selección especifica del agente que puede ser personalizada. Normalmente esta función es modificada por el programador para tomar en cuenta eventos con mayor prioridad para el agente dada la aplicación. Sólo en aplicaciones simples, donde puede asumirse que todos los eventos tienen la misma importancia para el agente, es recomendable utilizar la función de selección de eventos predefinida.El conjunto de eventos es implementado en Jason como una lista, los eventos nuevos son añadidos al final de la lista. Lo que la función de selección de eventos por defecto hace es simplemente devolver el primer elemento de la lista, teniéndose una cola de eventos.

- Paso 6 - Devolver todos los planes relevantes: ya que se ha seleccionado un evento, es necesario encontrar un plan que permita al agente actual de tal forma que pueda manejar el evento. Lo primero es encontrar, en la biblioteca de planes, todos los planes que son relevantes para el evento dado. Esto se logra devolviendo todos los planes de la biblioteca de planes del agente que tienen un evento disparador que puede ser unificado con el evento seleccionado.

- Paso 7 - Determinar los planes aplicables: los planes tienen un contexto que determina si un pan puede ser utilizado en un momento particular del tiempo, dada la información que el agente tiene en ese momento. Así, incluso si se han seleccionado todos los planes relevantes, no puede utilizarse cualquiera de ellos para manejar el evento actual. Se necesita seleccionar, de los planes relevantes, todos aquellos que son actualmente aplicables; esto es, se intenta seleccionar aquellos planes, que dadas las creencias actuales del agente, tienen más posibilidades de ser exitoso. Para lograr lo anterior, es necesario verificar si el contexto de cada plan relevante es verdadero; en otras palabras, si el contexto es una consecuencia lógica de la base de creencias del agente.

- Paso 8 - Seleccionar un plan aplicable: todos los planes aplicables seleccionados en el paso anterior son alternativas adecuadas para manejar el evento seleccionado. Esto significa que, hasta donde el agente sabe, cualquiera de esos planes son igual de buenos en el sentido de que el ejecutar alguno de ellos será suficiente para manejar el evento particular seleccionado en el ciclo de razonamiento. Consecuentemente l agente necesita escoger uno de esos planes aplicables y comprometerse a su ejecución. Comprometerse a ejecutar un plan significa que el agente tiene la intención de seguir el curso de acción determinado por el plan, de tal forma que pueda esperarse que el plan seleccionado terminé en algún momento en el conjunto de intenciones del agente. La selección de un plan en particular dentro del conjunto de planes aplicables que será incluido al conjunto de intenciones se lleva a cabo por otra función de selección, llamada función de selección de opción ('SO'). Cada uno de los planes aplicables pueden ser considerados una de varias opciones que el agente tiene con respecto a alternativas viables para manejar el evento seleccionado. Las metas que actualmente se encuentran en el conjunto de eventos representan los diferentes deseos de los cuales el agente puede escoger comprometerse, mientras que diferentes planes aplicables para una meta representan cursos de acción alternativos que el agente puede utilizar para conseguir una meta en particular. Así como con otras funciones de selección, SO es también modificable. El comportamiento por defecto de esta función es escoger el plan que aparece primero en la biblioteca de planes. El orden de los planes está determinado por el orden en que los planes son escritos en el código fuente del agente. Este ordenamiento puede ser útil si el evento disparador requiere de planes recursivos, en los cuales el programador sabe que el plan para detener la recursión debe aparecer primero, tal como se recomienda para Prolog.

  Cabe mencionar que sin importar cual sea el plan de la biblioteca de planes seleccionado para convertirse en intención, esta nueva intención es sólo una instancia (una copia) del plan. La biblioteca de planes no es modificada; es la instancia del plan la que es manipulada por el interprete.

- Paso 9 - Seleccionar una intención para continuar su ejecución: típicamente un agente tiene más de una intención en su conjunto de intenciones, cada una representa un diferente foco de atención. Estas intenciones se encuentran compitiendo por la atención del agente, lo que significa que cada una de ellas puede continuar ejecutándose, lo que será realizado en el próximo paso del ciclo de razonamiento. Sin embargo, en un ciclo de razonamiento, a lo más una fórmula de una de las intenciones es ejecutada. Consecuentemente, lo que debe hacerse antes de que el agente pueda actuar es escoger una intención en particular de entre aquellas que actualmente se encuentran en ejecución. Para lograr esto se utiliza una función de selección; a esta función se le llama función de selección de intención (SI). Como en casos anteriores, la idea es que tal decisión de selección es típicamente especifica para el agente. Para agentes complejos quizás sea necesario que el agente haga una selección informada basada en la urgencia de completar algunas de sus intenciones.

La función de selección de intenciones por defecto es un anillo 'round-robin'. Esto es, cada intención es seleccionada por turno, y cuando es seleccionada, sólo una fórmula del cuerpo del plan es ejecutada.

- Paso 10 - Ejecutar un paso de la intención: existen tres cosas principales que el agente realiza en cada ciclo de razonamiento: actualizar su información acerca del mundo y otros agentes, manejar uno de los muchos posibles eventos generados y entonces, como es de esperar, actuar sobre el ambiente (o de forma más general, seguir una de sus intenciones). Dada una intención en particular, es sencillo escoger que hacer, todo depende del tipo de fórmula que necesita ser ejecutado. Las fórmulas pueden ser: acciones del ambiente, metas de logro (sub-metas), metas de prueba, notas mentales y acciones internas. Cada uno de estos tipos puede modificar de forma diferente al agente, a otros agentes, al ambiente o al conjunto de intenciones actual.

Jason además brinda medios para que el programador extienda y modifique el ciclo de razonamiento mediante la definición de arquitecturas de agente. Para información más detallada y completa sobre Jason ver (Bordini y cols., 2007).

3.2. Ambientes Endógenos y CArtAgO

El medio ambiente de un sistema multi-agente juega un papel primordial, siendo éste la motivación de la acción del agente, recordador que el objetivo general de un agente es conseguir un estado deseable del medio ambiente. En muchas aplicaciones de sistemas multi-agente, el medio ambiente es el mundo real, esto sucede en robótica, control de sistemas de sensores, etc. El medio ambiente puede ser así mismo computacional, como es el caso de internet o una aplicación en especifico como podría ser un procesador de textos. En ocasiones es deseable crear un ambiente computacional para simular el mundo real u otro ambientes complejos, estas simulaciones permiten integrar la dinámica del sistema de una forma controlable y reproducible.
En sistemas multi-agente la creación de sistemas computacionales es una práctica común e importante, una definición inadecuada del ambiente puede provocar que los agentes se comporten de forma extraña aun cuando estén bien diseñados. Por lo anterior, incorporar la noción de ambiente en el momento de diseño del sistema es una buena práctica. Dada la necesidad de crear ambientes computacionales, Jason incorpora una serie de objetos Java con este fin. Sin embargo, los objetos java no proveen una abstracción adecuada del ambiente, resultando en ocasiones muy generales. Los objetos interactúan entre si mediante invocación de métodos sin que hagan referencia a acciones y percepciones; por el otro, los lenguajes orientados a agentes no definen el concepto de invocación de métodos, de forma que no tiene mucho sentido hablar de interacción entre agentes y objetos en términos de invocación de métodos. Es por esta razón que se han creado lenguajes de alto nivel, como es el caso de ELMS (Okuyama, Bordini, y da Rocha Costa, 2005), para diseñar e implementar ambientes computacionales para sistemas multi-agente. La programación de un sistema multi-agente (SMA) debería resumirse en la ecuación:

$SMA = Agnetes + ambiente$

Tradicionalmente se concibe al ambiente como algo exterior (exógeno) al agente, que éstos pueden percibir y modificar a partir de sus acciones y metas. En la concepción endógena del medio ambiente, éste forma parte del propio sistema multi-agente y es programable. Esta es la concepción que adopta la Ingeniería de Software orientada a agentes (AOSE por sus siglas en ingles), donde el ambiente es una abstracción de primera clase que encapsula funcionalidad y servicios que dan soporte a las actividades de los agentes. Estas funcionalidades deben servir para que los agentes logren sus metas en al menos tres niveles:

- *Contexto de despliegue*: se refiere a los recursos externos de software y hardware con los que el SMA puede interactuar: sensores, actuadores, bases de datos, servicios web, etc.
- *Capa de abstracción*: es una interfaz entre la representación a nivel agente y los detalles de bajo nivel presentes en el contexto de despliegue, de forma que el programador no necesite preocuparse por estos últimos.
- *Capa de mediación e interacción*: se refiere a explotar el ambiente para regular el acceso a recursos compartidos y para mediar la interacción entre agentes.

En un sistema multi-agente con ambiente endógeno, el ambiente es una parte programable del sistema, es ortogonal a éste pero esa fuertemente ligado a la parte de agente. En este tipo de sistemas, los agentes continúan siendo la abstracción principal para diseñar y programar las partes autónomas del sistema, en particular los componentes guiados por metas y tareas individuales y sociales. El ambiente se usa para diseñar y programar la parte computacional del sistema que ofrece funcionalidades a los agentes. Un ejemplo que pone de manifiesto la utilidad de la concepción de ambiente endógeno por encima de la concepción tradicional de ambiente es el siguiente. Considérese que es necesario implementar un pizarrón como medio de comunicación en un SMA, utilizando un ambiente tradicional no se tiene las estructuras adecuadas para dar soporte a objeto de este tipo, por lo que se terminaría optando por utilizar un agente que represente al pizarrón, sin embargo, esta solución es conceptualmente inadecuada ya que un pizarrón no es por definición un agente. Utilizando una concepción de ambiente endógeno, en cambio, el pizarrón puede implementarse como un recurso del ambiente accesible a los agentes a través de sus acciones y percepciones.

Un modelo computacional enfocado a la programación de sistemas multi-agente con ambientes endógenos debería tener las siguientes características:

- *Abstracto*: el modelo adoptado debe preservar el nivel de abstracción de los agentes. Esto es, los conceptos usados para programar la estructura y dinámica de los ambientes debe ser consistente con los conceptos usados para programar los agentes y su semántica. Ejemplos de esto incluyen las nociones de acción, percepción, evento, meta, tarea, etc.

- *Ortogonal*: el modelo debe ser lo más ortogonal posible con respecto a los modelos, arquitecturas y lenguajes adoptados para la programación de agentes; de forma que soporte naturalmente la ingeniería de sistemas heterogéneos.
- *General*: el modelo debe ser lo suficientemente expresivo y general como para desarrollar diferentes tipos de ambientes de acuerdo a diferentes dominios de aplicación y problemas, explotando el mismo conjunto básico de conceptos y constructores.
- *Modular*: el modelo debe concebir al ambiente como algo modular, evitando visiones de éste monolíticas y centralizadas.
- *Extensión dinámica*: el modelo debe soportar la construcción dinámica, remplazo y extensión de las partes del ambiente, en una perspectiva de sistema abierto.
- *Reutilización*: el modelo debe promover la reutilización de las partes de un ambiente en diferentes aplicaciones, contextos o dominios. Esto se enfatiza así mismo por la modularidad de éste.

CArtAgO (Ricci, Piunti, y Viroli, 2011) es un marco de trabajo computacional para la Programación de Ambientes basado en el meta-modelo de Agentes y Artefactos (ver figura 3.4) que satisface los requerimientos de programación de un ambiente endógeno. En CArtAgO el ambiente se concibe como un conjunto dinámico de entidades computacionales denominadas artefactos. Los artefactos representan recurso y herramientas que pueden ser utilizados y compartidos por los agentes que se encuentran en el mismo ambiente. Los artefactos tienen la característica de que pueden pertenecer a distintos espacios de trabajo, pudiéndose distribuir entre diversos nodos de cómputo de forma transparente, lo que vuelve a los artefactos atractivos en sistemas distribuidos. Desde el punto de vista del diseñador y el programador de SMA, la noción de artefacto es una abstracción de primera clase, el módulo básico para estructurar y organizar el ambiente, proveyendo una programación de propósito general y un modelo computacional para dar forma a la funcionalidad disponible para los agentes. Desde el punto de vista del agente, los artefactos son las entidades de primera clase que estructuran, desde un punto de vista funcional, el mundo computacional donde están situados. Los artefactos pueden ser creados, compartidos, usados y percibidos en tiempo de ejecución.

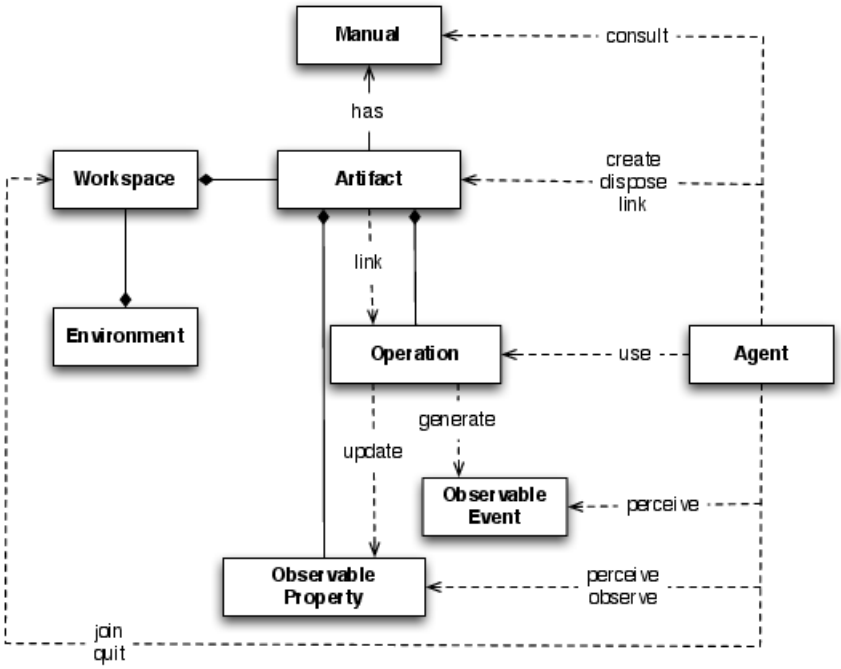

Figura 3.4.: Meta-modelo de agentes y artefactos

Para poner su funcionalidad a disposición de los agentes, un artefacto provee un conjunto de operaciones y otro de propiedades observables (Ver Figura 3.5). Las ope- raciones representan procesos computacionales, posiblemente de largo término, ejecutados dentro de los artefactos; que pueden ser disparados por agentes o por otros artefactos. El término interfaz de uso se refiere al conjunto de todas las operaciones representan variables de estado cuyos valores pueden ser percibidos por los agentes que están observando el artefacto. El valor de una propiedad observable puede cam- biar dinámicamente como resultado de la ejecución de una operación. La ejecución de una operación puede generar también señales perceptibles por los agentes. A diferencia de las propiedades observables, las señales pueden usarse para representar eventos observables no persistentes ocurridos dentro del artefacto. de un artefacto que están disponibles para un agente. Las propiedades observables representan variables de estado cuyos valores pueden ser percibidos por los agentes que están observando el artefacto. El valor de una propiedad observable puede cambiar dinámicamente como resultado de la ejecución de una operación. La ejecución de una operación puede generar también señales perceptibles por los agentes. A diferencia de las propiedades observables, las señales pueden usarse para representar eventos observables no persistentes ocurridos dentro del artefacto.

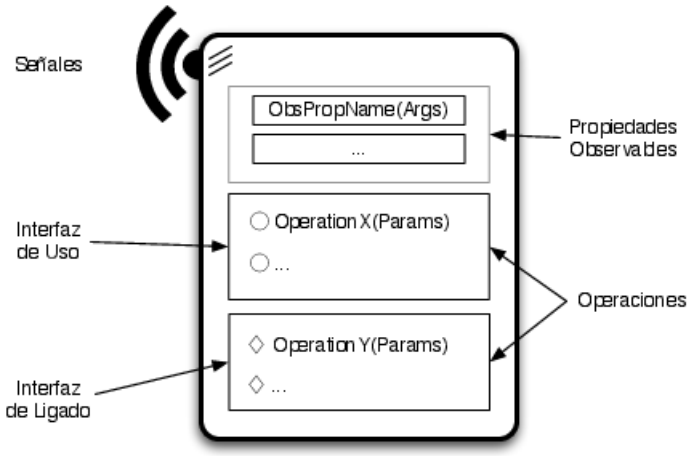

Figura 3.5.: Representación de un artefacto

Desde la perspectiva de un agente, las operaciones de un artefacto representan las acciones externas provistas por el ambiente a los agentes. Este es un aspecto central del modelo. De forma que en los ambientes basados en artefactos el repertorio de acciones externas disponible para un agente, aparte de aquellas acciones relacio- nadas con la comunicación, se define por el conjunto de artefactos en su ambiente. Esto implica que el repertorio de acciones puede ser dinámico dado que el conjunto de artefactos disponible puede ser cambiado dinámicamente por los agentes. Las propiedades observables y los eventos constituyen la percepción de un agente. En los lenguajes BDI como Jason, 2APL (Dastani, 2008) o GOAL (Bordinim, Dastani, Dix, y El Fallah Seghrouchni, 2009), las percepciones están relacionadas con el valor de las propiedades observables que pueden modelarse directamente dentro de un agente como creencias acerca del estado actual del ambiente. De hecho, para escalar con la complejidad de un ambiente basado en artefactos, un agente puede dinámicamente focalizar solo en los artefactos que le interesan.

Como principio de composición, los artefactos pueden ligarse entre si, de forma que un artefacto dispare la ejecución de operaciones de otro artefacto. Con este propósito, los artefactos exhiben una interfaz de ligado que análogamente a la interfaz de uso para los agentes, incluye el conjunto de operaciones que puede ser ejecutada por otros artefactos, una vez que dichos artefactos han sido ligados por un agente. La semántica de ejecución para las operaciones ligadas es la misma que la de las operaciones ejecutadas por los agentes: la solicitud de operación ejecutada por el artefacto que está ligando es suspendida hasta que la operación en el artefacto ligado ha sido ejecutada con éxito o fracaso. Las operaciones ligadas no son accesibles a los agentes, solo a artefactos ligados. El ligado permite la realización de ambientes distribuidos al facilitar el ligado de artefactos posiblemente situados en espacios de trabajo diferentes, situados en diferentes nodos de una red de cómputo. De esta manera los artefactos pueden ser tratados como componentes en el contexto de ingeniería del software orientada a componentes, asumiendo que las

interfases adecuadas para conectar los artefactos son provistas tal y como son requeridas.

Finalmente, un artefacto puede tener un manual en un formato legible por la computadora para ser consultado por los agentes. El manual contendría una descripción de las funcionalidad provista por el artefacto y la manera de explotar dicha funcionalidad. Esta característica ha sido concebida pensando en los Sistemas Abiertos compuestos por agentes inteligentes que dinámicamente deciden que artefactos usar de acuerdo a sus metas. De hecho, la noción de manual puede extenderse de los artefactos a los ambientes de trabajo. En este caso, el manual debe contener la descripción de los protocolos de uso que puede involucrar el uso de multiples tipos de artefactos.

4. Aprendizaje colaborativo de conceptos en SMA

En diversos problemas y aplicaciones, es necesario que los agentes, de forma individual, tomen decisiones basadas en información incierta o incompleta. Las conclusiones que pueden derivar a partir de su conocimiento directo puede ser insuficiente. Cuando la información está incompleta, es posible hacer algunas asunciones acerca de hechos o reglas desconocidas. Estas asunciones pueden tomar la forma de hipótesis que pueden ser utilizadas para completar el conocimiento del agente, sin embargo, dichas hipótesis deben poder ser revisadas, de tal forma que sea posible refinarlas (Bourgne, Maudet, y Pinson, s.f.).

En aprendizaje colaborativo de conceptos en sistemas multi-agente un grupo de agentes perciven individual y localmente el ambiente (mediante ejemplos), y cooperan para que se pueda llegar a un concepto general satisfactorio (hipótesis). De acuerdo a (Bourgne, Fallah-Seghrouchni, y Soldano, 2009) se deben tomar en cuenta al menos dos características cuando un grupo de agentes distribuidos se encuentran colaborando para lograr una tarea de aprendizaje en común (suponiendo que todos los agentes son confiables y están dispuestos a colaborar):

- La tarea de aprendizaje presumiblemente es incremental, esto es, el conjunto de entrenamiento no está dado a priori. En vez de ello, los ejemplos son colectados incrementalmente (y localmente) por los agentes
- La comunicación entre los agentes no puede estar garantizada. Esto puede ser debido al hecho de que los canales de comunicación son poco confiables, y/o existen restricciones en la topología de comunicación que impiden la comunicación entre ciertos agentes

Un ejemplo de aplicación de aprendizaje colaborativo puede ser una red distribuida de sensores, donde cada sensor (visto como un agente) tiene control sobre un área restringida, y necesita interactuar con sensores vecinos para poder analizar el comportamiento de alguna entidad desconocida. El reto para los investigadores de este campo es el diseñar mecanismos que sean eficientes, tomando en cuenta las necesidades altas de comunicación. En situaciones complejas, donde se involucra incertidumbre y comunicación no confiable, el aprendizaje se lleva a cabo generalmente con modelos numéricos, un ejemplo aplicación de dichos modelos puede encontrarse en (Stone, 2007).

4.1. SMILE

SMILE (Sound Multi-agent Incremental LEarning) (Bourgne y cols., 2007) es un método para realizar aprendizaje supervisado colaborativo de conceptos en sistemas multi-agente. Es en este trabajo en el cual la estrategia de aprendizaje aquí presentada se inspira, por lo tanto es menester presentar dicho método. Antes de presentar el método como tal, es necesario realizar una revisión de conceptos generales. En (Guerra-Hernández, El Fallah-Seghrouchni, y Soldano, 2005) se introduce una caracterización de aprendizaje en sistemas multi-agente de acuerdo al nivel de conciencia de los agentes. En el nivel 1, los agentes aprenden en el sistema sin considerar la presencia de otros agentes, excepto por las modificaciones que otros agentes hagan al ambiente. El nivel 2 implica interacción directa entre los agentes, intercambiando mensajes para mejorar el aprendizaje. El nivel 3 requiere que el agente considere las competencias de otros agentes, y a sabiendas de dichas competencias sea capaz de aprender mediante la observación del comportamiento de otros agentes.

SMILE es un método que se enfoca en el nivel 2 de aprendizaje, habiendo entonces interacciones entre los agentes en el proceso de aprendizaje. Se asume que cada agente es capaz de aprender incrementalmente de los datos que recibe, esto significa que cada agente puede actualizar su conjunto de creencias B para mantenerla consistente con el conjunto total de información K que ha recibido del ambiente o de otros agentes. En dicho caso, se dice que el agente es $a-consitente$. En este caso, el conjunto de creencias B representa conocimiento hipotético que puede ser revisado, y el conjunto de información K representa conocimiento verdadero, que consiste de observaciones y hechos no revisables. Se asume que al menos un conjunto B_c de creencias es común para todos los agentes y debe permanecer de esa forma. Consecuentemente, una actualización de este conjunto común B_c por un agente r debe provocar una actualización de B_c para toda la comunidad de agentes. El proceso de actualización de las creencias de la comunidad cuando uno de sus miembros obtiene nueva información puede ser definido como el proceso de mantenimiento de consistencia que asegura que cada agente de la comunidad permanecerá $mas-consitente$. Este proceso de mantenimiento de $mas-consitencia$ de un agente que obtiene nueva información le da el rol de aprendiz e implica comunicación otros agentes que actúan como críticos. Los agentes pueden ser aprendices o críticos, intercambiando estos roles en el momento adecuado.

En (Bourgne y cols., 2007) se llevan a cabo las siguientes definiciones formales. Se representa un sistema multi-agente como un conjunto de agentes $r_1, ..., r_n$. Cada agente r_i tiene un conjunto de creencias B_i que consiste de todo el conocimiento revisable que posee. Parte de ese conocimiento debe ser compartido con otros agentes. La parte de B_i que es común a todos los agentes se denota como B_c. Esta parte en común produce una dependencia entre los agentes. Si un agente r_i actualiza su conjunto de creencias B_i a B'_i, cambiando en el proceso B_c por B'_c todos los demás agentes r_k deben entonces actualizar su conjunto de creencias B_k por B'_k, de tal forma que $B'_c \subseteq B'_k$. Cada agente

r_i almacena cierta información K_i. Una propiedad de consistencia $Cons(B_i.K_i)$ puede ser verificada por el agente mismo entre sus creencias B_i y su información K_i. Como se mencionó antes, B_i representa conocimiento revisable, mientras que K_i representa hechos observados, considerados como verdaderos, y que posiblemente pueden contradecir B_i. Un agente r_i es $a - consitente$ si y sólo si $Cons(B_i, K_i)$ es verdadero.

También es deseable definir alguna noción de consistencia del sistema multi-agente como un todo que dependa de los conjuntos de creencias e información de sus elementos constituyentes. La consistencia de un agente r_i con respecto de su conjunto de creencias B_i y su propio conjunto de información K_i junto con todos los conjuntos de información $K_i, ..., K_n$ de los otros agentes se considera como $mas - consistencia$. Un agente r_i es $mas - consistente$ ssi $Cons(B_i, K_i \cup K)$ es verdadera. K es el conjunto de información de todos los agentes del sistema. La consistencia global del SMA es entonces la $mas - consistencia$ de todos los agentes. Un SMA $r_1, ..., r_n$ es consistente ssi todos sus agentes r_i son $mas - consistente$.

Para definir las propiedades requeridas con que debe cumplir un mecanismo de revisión M que actualiza a un agente r_i cuando éste obtiene una pieza de información k, es necesario suponer lo siguiente:

- La actualización es siempre posible, esto es, un agente puede modificar siempre su conjunto de creencias B_i para recuperar $a - consitencia$. Se dice entonces que cada agente es $localmente\ eficiente$
- Considerando dos conjuntos de información $Cons(B_i, K_i)$ y $Cons(B_i, K_2)$, se tiene también $Cons(B_i, K_1 \cup K_2$. Esto es, la $a - consitencia$ es aditiva.
- Si una pieza de información k concerniente al conjunto común B_c es consistente con un agente, es consistente con todos los agente: para todo par de agentes (r_i, r_j) tal que $Cons(B_i, K_i)$ y $Cons(B_j, K_j)$ son verdaderos, se tiene que para toda pieza de información k: $Cons(B_i, K_i \cup k)$ ssi $Cons(B_j, K_j \cup k)$. En tal caso se dice que el SMA es coherente.

M es visto como un mecanismo de aprendizaje incremental y representado como una aplicación que cambia B_i a B_i'. Un mecanismo de actualización M es $a - consitente$ ssi por cualquier agente r_i y cualquier pieza de información k que alcance a r_i, la $a - consitencia$ de este agente es preservada. En otras palabras, ssi: $r_i(B_i, K_i)a - consistente \Rightarrow r_i(B_i', K_i')a - consistente$, donde $B_i' = M(B_i)$ y $K_i' = K_i \cup k$ es el conjunto de toda la información de otros agentes del SMA.

Un mecanismo de actualización M_s es $mas - consistente$ sii para todos los agentes r_i y para todas las piezas de información k que alcanzan a r_i, la $mas - consistencia$ de este agente es preservada. En otras palabras ssi: $r_i(B_i, K_i, K)mas - consistente \Rightarrow r_i(B_i', K_i', K)a - consistente$, donde $B_i' = M_s(B_i)$ y $K_i' = K_i \cup k$ y $K = \cup K_j$ es el

conjunto de toda la información del SMA.

Cuando el mecanismo de $mas - consistencia$ es aplicado por un agente que obtiene una nueva pieza de información, un efecto colateral deseable del mecanismo es que otros agentes permanezcan $mas - consistentes$ después de cualquier modificación de la parte común de B_c, esto es, el mismo SMA debe volver a ser consistente. A esta propiedad se le conoce como $mas - consitencia\ fuerte$ y se define formalmente de la siguiente forma. Un mecanismo de actualización M_s es $mas - consitente\ fuerte$ ssi: M_s es $mas - consistente$, y la aplicación de M_s por un agente preserva la consistencia del SMA.

La idea general del mecanismo de aprendizaje SMILE, es que dado que la información está distribuida entre todos los agentes del SMA, debe haber algún tipo de interacción entre el agente aprendiz y los otros agentes todo englobado en un mecanismo $mas - consitente\ fuerte\ M_s$. Para asegurar la $mas - consitencia$ del mecanismo, M_s debe consistir de una aplicación reiterativa, por parte del agente aprendiz r_i, de un mecanismo interno de $a - consistencia\ M$. Este mecanismo es lanzado por el agente r_i en el momento que recibe una pieza de información k que quebrante la $mas - consitencia$. Se denota a $M(B_i)$ como el conjunto de creencias del agente Aprendizaje r_i después de una actualización, B'_c como la parte común modificada por r_i, y B'_j como el conjunto de creencias modificadas de otro agente r_j, modificación inducida por las modificaciones de su parte común B_c en B'_c.

Una interacción $I(r_i, r_j)$ entre el agente aprendiz r_i y otro agente r_j que actúa como crítico consta de los siguientes pasos:

- El agente r_i envía la actualización B_c de la parte común de sus creencias. Habiendo aplicado su mecanismo de actualización, r_i es $a - consitente$.
- El agente r_j revisa la modificación B'_j de sus creencias, inducida por la actualización de B_c. Si esta modificación preserva su $a - consitencia$, r_j adopta la modificación.
- El agente r_j envía, o bien la aceptación de B'_c o bien su rechazo. El rechazo va acompañado junto con una o más piezas de información k', tal que $Cons(B'_j, k')$ es falso.

Así, una iteración de M_s se compone de:

- la recepción por parte del agente aprendiz r_i de una pieza de información y la actualización $M(B_i)$ recuperando su $a - consistencia$
- un conjunto de interacciones $I(r_i, r_j)$, en los cuales varios agentes críticos pueden participar. Si al menos una pieza de información k' es transmitida a r_i, la adición k' necesariamente hará que r_i sea $a - consitente$ y una nueva iteración ocurrirá

Este mecanismo M_s termina cuando ningún agente puede proveer tal pieza de información k'. Cuando este es el caso, la $mas - consitencia$ del agente aprendiz r_i es restaurada.

En el mecanismo M_s descrito, el agente aprendiz es el único que recibe y memoriza información durante la ejecución. Esto asegura que M_s termina. La pieza de información transmitida por otros agentes y memorizada por el aprendiz son redundantes, ya que se encuentran presentes en el SMA, más precisamente en la memoria del agente crítico que las transmite.

El mecanismo anteriormente descrito, no indica explícitamente el orden o el alcance de las interacciones. Es en este punto donde se da lugar a propuestas. En Smile (Bourgne y cols., 2007) las interacciones son secuenciales y síncronas, todo desarrollado en un ambiente local. Mientras tanto en (Bourgne y cols., 2010) se opta por interacciones concurrentes o paralelas asíncronas que pueden desarrollarse en un ambiente distribuido.

5. KDD y minería de datos

En los últimos tiempos, nuestra habilidad para reunir y almacenar datos ha superado significativamente a nuestra habilidad de analizar y extraer conocimiento de este continuo flujo de datos. Los métodos de análisis de datos tradicionales que requieren de personas para procesar grandes cantidades de datos son totalmente inaplicables. El campo de descubrimiento de conocimiento y minería de datos (KDD por sus siglas en ingles) ha surgido para compensar esta deficiencia. El descubrimiento de conocimiento en bases de datos denota el proceso complejo de identificar validos, noveles, potencialmente útiles y en ultima instancia entendibles, patrones en los datos (Fayyad, Piatetsky-Shapiro, Smyth, y Uthurusamy, 1996). La minería de datos se refiere a un paso en particular del proceso KDD. De acuerdo a una definición aceptada (Fayyad y cols., 1996), la minería de datos está constituida por un conjunto particular de algoritmos (métodos) que, bajo limitaciones aceptables de eficiencia computacional, producen una enumeración particular de patrones (modelos) a partir de los datos.

A través de la literatura sobre el tema, los términos KDD y minería de datos son muchas veces utilizados de forma indistinta. Una convención aceptada establece que el proceso de descubrimiento de conocimiento consiste de tres tareas (Blockeel y De Raedt, 1998). La primera tarea es adaptar el formato original de los datos para que éstos tomen el formato de entrada del algoritmo de minería de datos, a esta tarea se le llama pre-procesamiento. En el pre-procesamiento caen sub-tareas como son: discretizado de los datos, tratado de valores incompletos, selección de registros, creación sintética de registros, modificación de atributos, etc. Una vez que los datos se encuentran formateados, uno o más algoritmos son aplicados para extraer patrones, regularidades o leyes generales de los datos, esta es la fase propiamente llamada minería de datos. Contando con los resultados del proceso de minería de datos, éstos necesitan ser traducidos a un formato inteligible, a este proceso se le conoce como post-procesamiento de los resultados. Algunas técnicas de post-procesamiento son, por ejemplo, la creación de modelos gráficos, producción de reglas lógicas, interpretación estadística de los resultados, etc.

En un contexto de base de datos relacional, una tarea típica de minería de datos es explicar y predecir el valor de algún atributo dada una colección de tuplas con valores conocidos para dicho atributo. Una forma de lograr lo anterior es mediante la utilización de algoritmos de minería de datos. Un conjunto de datos, extraído de un dominio en particular, es utilizado como conjunto de entrenamiento para un algoritmo de aprendizaje que computa una expresión lógica, la descripción de un concepto, un modelo descriptivo, o un clasificador, que más tarde puede ser utilizado para predecir (por diversas razones particulares a la aplicación) el valor de un atributo especificado para ciertos registros

cuyo valor de este atributo es desconocido.

El campo de minería de datos ha logrado progresos substanciales a lo largo de los últimos tiempos, y numerosos algoritmos han emergido, desde aquellos basados en modelos estocásticos hasta aquellos basados en representaciones simbólicas puras tales como reglas y árboles de decisión. Desde la pasada década, la minería de datos logró moverse de un ambiente puramente experimental y de laboratorio a un ambiente de aplicación con valor comercial (Mitchell, 1997). Los algoritmos de minería de datos han sido empleados exitosamente en diversas aplicaciones tales como diagnostico de enfermedades del corazón (Detrano y cols., 1989), predicción de niveles de glucosa en pacientes diabéticos (Lehmann y Deutsch, 1998), detección de fraudes en tarjetas de crédito (Stolfo, Fan, Lee, Prodromidis, y Chan, 1997), etc.

5.1. Árboles de decisión

En teoría de grafos un árbol es un grafo acíclico dirigido donde cada nodo puede tener sólo un padre (exceptuando la raíz) (Neapolitan, 2004). En un árbol de decisión cada nodo representa un atributo y cada arco el valor del atributo de donde parte (ver figura:5.1). Las hojas representan valores de clase. Un árbol de decisión es un espacio de hipótesis, cada camino que parte de la raíz es una hipótesis y los caminos se bifurcan a partir de disyunciones. Por lo tanto, es sencillo realizar una clasificación a partir de un árbol de decisión, basta con recorrer el camino que se corresponde con los valores de la instancia que se desea clasificar. Para realizar clasificaciones acertadas, es necesario que el árbol de decisión sea bueno, existen diversos métodos de construcción de árboles de decisión (a este proceso también se le conoce como inducción), siendo ID3 uno de los métodos historicamente más influyentes.

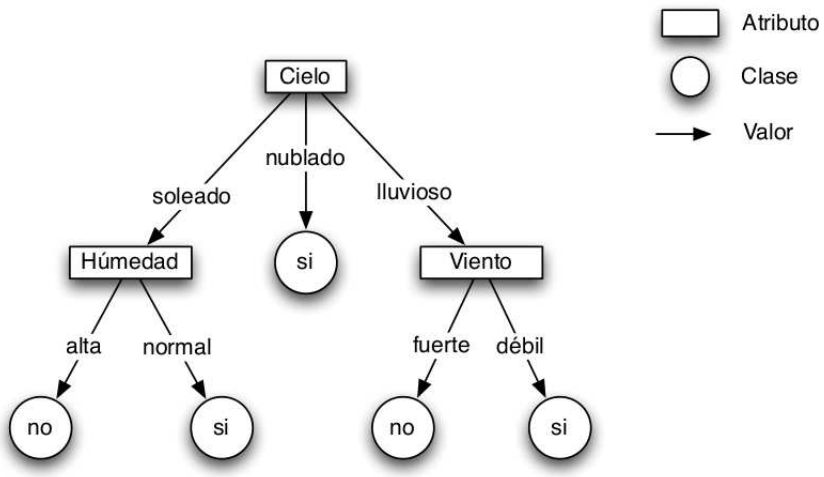

Figura 5.1.: Árbol de decisión de la conocida base de datos Tenis

ID3 es un algoritmo que permite crear modelos de clasificación a partir de un conjunto de datos, produce un árbol de decisión del cual es posible extraer reglas. La parte central del algoritmo ID3 es la construcción del árbol de decisión, para construir el árbol de decisión se sigue el siguiente procedimiento (en pseudo-código):

- $ID3(D) -> A$. Donde D es un conjunto de datos y A es el árbol de decisión resultante

1. Si se ha alcanzado un condición de paro, regresar una hoja (clase más común)
2. Escoger un atributo Att_i de D para realizar la partición
3. Crear particiones $[D_1, D_2, .., D_n], D_i \subseteq D$ donde cada partición D_i contiene valores iguales de Att_i , se elimina Att_i de cada D_i
4. Para cada D_i hacer la llamada recursiva $ID3(D_i)$ para obtener $[A_1.A_2., .., A_n]$ sub-árboles
5. Crear nodo A a partir de Att_i escogido en el paso 2 y ligar A con cada A_i poniendo como arco el valor de Att_i que corresponde
6. Regresar A

Tradicionalmente para ID3 la condición de paro es que todas las instancias pertenecientes a la partición actual sean de la misma clase. Otra condición de parada obvia es que no queden más atributos para crear particiones (recuerden que cada vez que se particiona se elimina un atributo). Otra condición puede ser que se ha alcanzado un porcentaje donde las instancias de la partición pertenecen a la misma clase. Puede haber

métricas más complicadas para determinar esta condición.

El paso 2 en el algoritmo anterior es una de los puntos más críticos del algoritmo, este punto es el que mayormente determina si el árbol de decisión es bueno o malo. Para determinar el atributo de partición ID3 utiliza una métrica conocida como "Ganancia de Información", esta métrica a su vez se basa en otra que se conoce como "Entropía". La idea general es que ID3 escoge los atributos que maximicen la información, esto quiere decir aquellos atributos que dividan de una forma más pura la base de datos, esto es que las particiones sean lo más homogéneas posibles en cuanto al valor de clase.

El algoritmo original de ID3 tiene la siguientes limitaciones generales:

- Sólo acepta atributos cuyos valores son nominales
- No integra técnicas de post-podado

El algoritmo C4.5 (Quinlan, 1993) es una mejora de ID3 que integra discretización automática basada en particionamiento binario, gain ratio como métrica de elección de candidatos, tratamiento de datos faltantes, y por último post-poda. En el trabajo presente se utiliza la implementación de C4.5 implementada en WEKA llamada J48.

La poda responde a la problemática que se plantea a continuación. Dado que el árbol que resulta de un proceso de inducción es el que mejor se ajusta al conjunto de datos, posiblemente no sea una buena generalización y por lo tanto cuando se clasifiquen datos nuevos habrá instancias mal clasificadas. Para prevenir este sobre ajuste se utiliza la poda. Los árboles de decisión son usualmente simplificados por el descarte de uno o más sub-árboles siendo remplazados por hojas. C4.5 permite el reemplazo de un sub-árbol por sus ramas o una hoja. La poda puede realizarse o bien al momento de inducir el árbol (pre-poda) o bien al terminarse el proceso de inducción (post-poda). El algoritmo de post-poda de Quinlan (Quinlan, 1993) empieza de abajo hacia arriba del árbol y examina cada nodo no hoja del subárbol. Si el reemplazo del subárbol con una hoja, o con su rama más frecuentemente usada lleva a una tasa de error más baja, entonces se poda el árbol recordando que la tasa de error predicha para todos los árboles que lo incluyen se verán afectados. El error estimado para un subárbol es la suma ponderada de los errores estimados para cada hoja. El error estimado para cada nodo se calcula como: $e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \middle/ \left(1 + \frac{z^2}{N}\right)\right)$

Donde:

- z es un valor constante que depende de un parámetro c que sirve para establecer un intervalo de confianza. Si c = 25 % entonces $z = 0{,}69$
- f es el error sobre los datos de entrenamiento
- N es el número de instancias cubiertas por una hoja

Los árboles de decisión pueden ser así mismo utilizados en problemas representados en términos de lógica de primer orden. Los métodos tradicionales de aprendizaje proposicional no funcionan en un contexto de primer orden, es por ello que surgen alternativas similares a las proposicionales, tal es el caso de TILDE (Blockeel y De Raedt, 1998). TILDE es un algoritmo utilizado para inducir árboles lógicos de decisión, utiliza un procedimiento muy similar al de ID3 pero adaptado para lidiar con los problemas que la lógica de primer orden introduce.

Los árboles de decisión también pueden ser extendidos para integrar probabilidades. En (Provost y Domingos, 2003) se hace un análisis sobre una versión modificada de C4.5 que integra probabilidades y se llega a la conclusión de que el modelo propuesto es bueno siempre y cuando no se aplique ningún tipo de podado y se integre algún tipo de corrección de probabilidades en las hojas como es la corrección de Laplace. Tal como existe la problemática de migrar modelos de aprendizaje proposicional a términos de lógica de primer orden, algo similar ocurre con la migración de métodos proposicionales probabilistas a un esquema de primer orden. El área de Aprendizaje Relacional Estadístico (Statical Relational Learning) (Kersting, 2005) se encarga precisamente de lidiar con el problema anteriormente mencionado. En (Fierens, Ramon, Blockeel, y Bruynooghe, 2005) se lleva a cabo una evaluación de una variante del algoritmo de TILDE para integrar probabilidades, esta variante es probada bajo distintas métricas y condiciones de paro. En el artículo anteriormente citado se llega a la conclusión de que la variante de TILDE evaluada da buenos resultados con los parámetros de configuración adecuados. En dicho artículo también da la impresión de que la variante sólo funciona bien en casos donde se tienen muchos ejemplos de entrenamiento, sin embargo, en el artículo se omitió la introducción de corrección de probabilidades en las hojas lo cual da lugar a probar con bases de datos más pequeñas y algún tipo de corrección como la de Laplace.

5.2. Minería de datos distribuida (DDM)

Tradicionalmente los algoritmos de minería de datos asumen que los datos se encuentran centralizados, residiendo en memoria, y estáticos. Sin embargo, esta asunción no es válida en los tiempos modernos, donde internet y las redes de cómputo en general son ubicuas. En estos ambientes distribuidos la minería de datos enfrenta dos grandes retos (Zeng y cols., 2012): En primer lugar, los datos son generados tan rápido que no es posible procesarlos, incluso cuando se cuenta con súper computadoras. En segundo lugar, los datos son almacenados en múltiples localidades, por lo que el costo de transmisión para centralizar todos los datos es proporcional a la cantidad de éstos. Las limitaciones de ancho de banda y preocupaciones de privacidad son otros de los factores que impactan en la centralización de datos.

Para resolver los problemas antes mencionados, la minería de datos distribuida (DDM desde ahora) ha surgido como un área de investigación activa. DDM se ha popularizado

en los últimos tiempos gracias a que el mercado de negocios inteligentes crece rápidamente, siendo una de las áreas más remuneradas de la industria de software actual (Zeng y cols., 2012).

DDM existe la asunción de que o bien los datos se encuentran distribuidos o bien el cómputo es distribuido. La aproximación de DDM puede ser utilizada en súper computadoras paralelas, redes P2P, redes de sensores, etc. Bajo ciertas circunstancias, el proceso de DMM puede tener restricciones de privacidad, comunicación, y recursos computacionales. Así mismo, DMM es un proceso que involucra la aplicación de algoritmos específicos, esto vuelve difícil el tener una plataforma general de algoritmos, cada aplicación tiene diferentes objetivos y preocupaciones. Las técnicas de DMM pueden ser categorizadas en los siguientes tipos (Zeng y cols., 2012):

- Clusters centrales VS peer-to-peer: el cluster central tiene un coordinador. El coordinador divide el trabajo entre múltiples computadoras. Este esquema es fácil de emplear y configurar, sin embargo, es difícil determinar la forma óptima de división de trabajo, así mismo. Este esquema también sufre del problema de único punto de fallo, Si el cluster central falla, todo el proceso falla. El esquema requiere de ambientes estables y es más usualmente utilizado en ambientes locales de súper computadoras. Por el contrario, las redes P2P no requieren de un coordinador central, cada nodo puede ser visto como cliente y servidor. Cada nodo de la red recibe su partición de datos o trabajo independiente. Dado que el esquema P2P es descentralizado, cada localidad tiene una vista limitada del sistema. Esta visión limitada enfatiza la seguridad global del sistema. Así mismo, algoritmos ligeros pueden ser transmitidos a través de la red en contra posición a grandes cantidades de datos.

- Modelo único VS Meta-Aprendizaje: en la aproximación de modelo único se reparte el proceso de minería de datos en diversas localidades, se crean modelos de aprendizaje local que más tarde son reunidos en una localidad central. Una localidad coordinadora reúne los resultados intermedios de cada sitio para crear un resultado final. Por otra parte, Meta-Aprendizaje (Prodromidis, Chan, y Stolfo, 2000), vagamente definido como aprendizaje a partir de conocimiento aprendido, es otra técnica que tiene por objetivo lidiar con el problema de crear un modelo global a partir de bastas e inherentemente distribuidas bases de datos. En Meta-Aprendizaje se computan un número de modelos independientes de forma paralela, uno por cada localidad, sin la necesidad de compartir los datos que almacena la localidad. Este tipo de sistemas es más flexible para seleccionar y combinar diferentes modelos de minería de datos. Un tratamiento más afondo de Meta-Aprendizaje se presenta en la sección 5.2.1

- Datos homogéneos VS datos heterogéneos: en un sistema con una base de datos relacional distribuida, los datos pueden estar almacenados en distintos sitios. Si cada sitio cuenta con la totalidad de las tablas relacionales, definidas y estructuradas de la misma forma en todos los sitios, entonces se dice que los datos son homogéneos.

Si por el contrario, en cada sitio existen diferencias en las tablas, ya sean tablas con formatos diferentes pero que son utilizadas para los mismos fines o bien tablas de más o de menos, entonces se tiene un problema de datos heterogéneos. Lidiar con bases de datos heterogéneas no es solo un problema de unificación de formatos, como en un principio podría pensarse, sino también es un problema de unificación de semántica, cada localidad puede tener su propia forma de describir un problema y unificar esas distintas descripciones representa un reto.

5.2.1. Meta-Aprendizaje

Meta-Aprendizaje es una técnica que busca computar clasificadores de alto nivel llamados meta-clasificadores, estos meta-clasificadores integran múltiples clasificadores computados de forma separada sobre distintas bases de datos (Prodromidis y cols., 2000). El Meta-Aprendizaje es una técnica que lidia con el problema de computar un clasificador global a partir de bases de datos grandes e inherentemente distribuidas. El objetivo del Meta-Aprendizaje es computar un número de clasificadores independientes al aplicar algoritmos de aprendizaje sobre una colección de bases de datos independientes y distribuidas, todo de forma paralela. A cada clasificador creado localmente se le denomina "clasificador base". Estos clasificadores base son reunidos y combinados en una localidad central mediante otro proceso de aprendizaje. En este punto el proceso de Meta-Aprendizaje busca computar un meta-clasificador, éste trata de integrar en alguna medida los clasificadores aprendidos de forma separada, de tal forma que se intente mejorar el poder predictivo de éstos. Para generar el meta-clasificador, tres técnicas generales pueden ser utilizadas (Prodromidis y cols., 2000):

- Voto: cuando surge una petición de clasificación, cada clasificador base emite un voto (clase predichas), la predicción mayoritaria gana. Existen variaciones de este método que utilizan votos ponderados.

- Arbitraje: un clasificador arbitro es aprendido a partir de un conjunto de ejemplos de entrenamiento especialmente escogido. El arbitraje se lleva a cabo basado en las predicciones de los clasificadores locales y el arbitro. Utilizando una regla de arbitraje, un con seso acerca de la clase solicitada puede ser logrado. La regla de arbitraje puede ser tan simple como en caso de empate dar prioridad a la predicción del arbitro. Este método parte de la idea de que el arbitro es un juez objetivo cuya predicción es seleccionada si los clasificadores participantes no pueden alcanzar una decisión en consenso.

- Combinadores: Generan meta-clasificadores basados en meta-datos (que es conocimiento acerca de cómo se comportan los clasificadores) creados por los clasificadores base. Así, si por ejemplo, cuando se tienen dos clasificadores que al predecir la misma clase siempre están en lo correcto (relativo a un conjunto de validación particular), este simple hecho puede llevar a una herramienta predictiva poderosa.

5.3. Minería de datos distribuida basada en agentes

Los sistema multi-agente (SMA) a menudo tratan con aplicaciones complejas que requieren de planteamientos distribuidos. En muchas aplicaciones, el comportamiento individual y colectivo de los agentes depende de los datos observados a partir de fuentes distribuidas. En ambientes distribuidos típicos, analizar datos distribuidos es un problema nada trivial debido a muchas limitaciones posibles, como son el ancho de banda, privacidad de los datos, distribución de los nodos de cómputo, etc. El campo de minería de datos distribuida ofrece diversas soluciones algorítmicas para desarrollar diferentes operaciones de análisis de una forma distribuida que presta especial cuidado a las limitaciones de recursos (Da Silva, Giannella, Bhargava, Kargupta, y Klusch, 2005). Dado que un SMA es también un sistema distribuido, combinar minería de datos distribuida y SMA es atractivo.

Los agentes en un SMA necesitan ser pro-activos y autónomos. Los agentes perciben su ambiente, razonan dinámicamente sobre acciones basadas en las condiciones del ambiente, e interactúan entre ellos. En algunas aplicaciones el conocimiento de los agentes que guían su razonamiento y acción depende de la teoría de dominio existente. Sin embargo, en muchos dominios complejos este conocimiento es el resultado de la salida de un análisis empírico de datos en conjunto con conocimiento de dominio preexistente. El análisis escalable de datos puede requerir de minería de datos avanzada para detectar patrones ocultos, construcción de modelos predicativos, identificación de outliers, entre otros. En sistemas multi-agente este conocimiento es usualmente colectivo. Esta 'inteligencia' colectiva de un SMA debe desarrollarse por conocimiento del dominio distribuido y análisis de datos distribuidos observados por diferentes agentes. Tal análisis de datos distribuidos puede ser un problema no trivial cuando la tarea no puede descomponerse por completo y los recursos de cómputo se encuentran limitados por diversos factores tales como suministro de energía, conexión de banda ancha pobre, y datos privados, entre otros.

Los sistemas de minería de datos distribuida son sistemas complejos que se enfocan en la distribución de recursos en red y en procesos de minería. El núcleo de un sistema de minería de datos distribuida es la escalabilidad y la flexibilidad que se tenga para alterar la configuración de éste. De esta forma, diseñar sistemas de minería de datos distribuida es enfrentarse con diversos problemas desde el punto de vista de ingeniería de software, tales como reusabilidad, extensibilidad y robustez (V. S. Rao, 2009). Por esas razones, las características que exhiben los agentes los hacen deseables para el desarrollo de sistemas de minería de datos distribuida. Los sistemas multi-agente inherentemente tienen una característica de descentralización que se ajusta muy bien a los requerimientos de sistemas de minería de datos distribuida. Los agentes auto monos pueden ser ocupados como unidades de cómputo que ejecutan múltiples tareas basadas en una configuración dinámica. Como la naturaleza de un SMA es descentralizada, cada agente tiene una vista limitada del sistema. Esta limitación permite mayor seguridad ya que los agentes no

necesitan observar partes irrelevantes para su trabajo. De esta forma, los agentes pueden ser programados de la forma más compacta posible, en la cual agentes ligeros pueden ser transmitidos a través de la red, evitándose la transmisión de grandes cantidades de datos.

Ser capaz de transmitir agentes de un host a otro permite una organización dinámica del sistema. Por ejemplo, el agente a_1, localizado en el sitio s_1, posee el algoritmo alg_1. La tarea t_1 del sitio s_2 necesita explorar los datos utilizando $alg1$. En este caso, transmitir a_1 a s_2 es mejor que transmitir todos los datos de s_2 a s_1 donde $alg1$ está disponible. La seguridad en este tipo de sistemas es un problema crítico. Los modelos de seguridad rígidos que intentan preservar la seguridad pueden de hecho afectar la escalabilidad del sistema. Los agentes ofrecen soluciones alternativas a este problema ya que pueden viajar por el sistema. En (Walter, Battiston, y Schweitzer, 2008) y (Gorodetski y Kotenko, 2002) puede encontrarse mayor información relacionada con minería de datos distribuida basada en agentes y seguridad.

Al construir un sistema de minería de datos distribuida basada en agentes se consideran tres aspectos fundamentales (V. S. Rao, 2009): interoperabilidad, configuración dinámica del sistema y aspectos de desempeño. La interoperabilidad se refiere a la colaboración de los agentes en el sistema y a las interacciones externas que le permiten a un agente integrarse al sistema. La arquitectura del sistema debe ser lo suficientemente abierta y flexible para permitir protocolos de comunicación, políticas de integración y directorio de servicios. Una tarea de minería de datos puede requerir de varios agentes y fuentes de datos, la configuración dinámica le permite a cada agente cambiar su rol, de tal forma que pueda cambiarse el algoritmo de minería de datos que utiliza o la base de datos que tiene asignada, así como moverse entre las distintas localidades de ser necesario. Por último, el desempeño se ve afectado principalmente por la distribución de datos, se debe determinar que tareas son paralelas y considerar los problemas de control de concurrencia que puedan surgir.

De acuerdo a (V. S. Rao, 2009), un sistema de minería de datos basada en agente puede ser generalizado como un conjunto de componentes. Dichos componentes se encuentran el la figura 5.2. Estos componentes básicos pueden categorizares en dos grupos, o bien son de petición o bien son de respuesta. Dichos componentes son los siguientes:

- Datos: es la capa fundamental de interés. En ambientes distribuidos, los datos pueden ser presentados de diversas formas, tales como bases de datos relacionales, streams de datos, páginas web, etc., en los cuales el propósito de los datos varía.
- Comunicación: el sistema elige los recursos relacionados a partir del directorio de servicios, el cual mantiene una lista de fuentes de datos, algoritmos, esquemas de datos, tipos de datos, etc. Los protocolos de comunicación pueden variar dependiendo de la implementación del sistema, tales como cliente-servidor, peer-to-peer, etc.
- Presentación: la interfaz de usuario interactúa con el usuario para simplificar el sis-

tema, brindando mensajes amigables que permiten una interacción sencilla. Cuando un usuario realiza la petición de una operación de minería de datos a través de la interfaz de usuario, los siguientes componentes están involucrados.

- Optimización de consulta: un optimizador de consultas analiza una petición para determinar el tipo de operación de minería de datos y elige los recursos apropiados para dicha petición. También determina si es posible paralelizar la tarea, dado que los datos están distribuidos y pueden ser procesados en paralelo.
- Plan de descubrimiento: un planificador almacena sub-tareas con junto con recursos relacionados. En este punto, agentes mediadores juegan un rol importante en la coordinación de múltiples unidades de cómputo.
- Descubrimiento de conocimiento: también conocido como minería, ejecuta el algoritmo requerido por la tarea para obtener conocimiento de una fuente de datos especifica. Es posible que exista una transformación previa en cada localidad para poder adaptar los datos.
- Consolidación de conocimiento: para poder presentarle al usuario unos resultados compactos y significativos, es necesario normalizar el conocimiento obtenido de las diversas fuentes. El componente involucra metodologías complejas para combinar conocimiento o patrones a partir de sitios distribuidos.

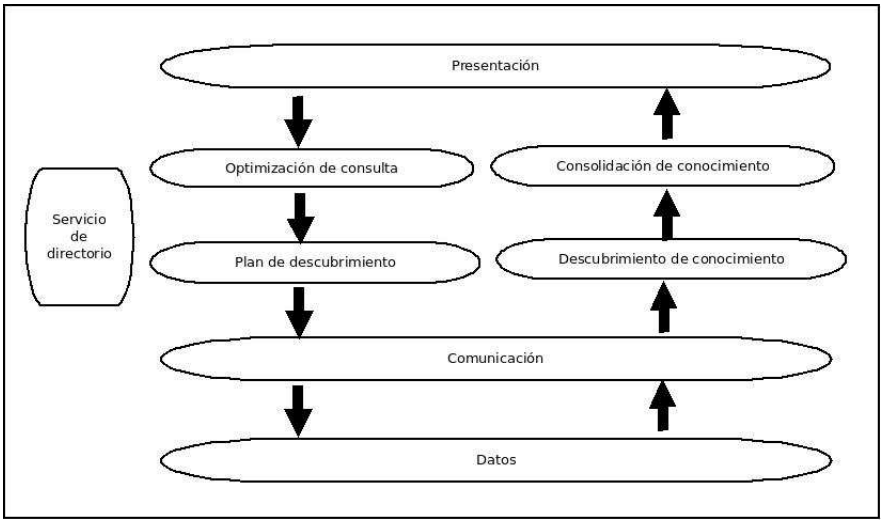

Figura 5.2.: Componentes de un sistema de minería de datos distribuida basado en agentes

5.4. Evaluación de modelos de clasificación

El saber si un método de clasificación arroja buenos resultados sólo puede ser comprobado en la práctica en casos reales, pero es inconcebible aplicar un método de clasificación en una situación real si ni siquiera se tiene algún grado de confianza en dicho método. La evaluación de modelos de clasificación lidia precisamente con este problema. Las evaluaciones de estos modelos normalmente consideran la certeza de clasificación como medida de desempeño. En esquemas de evaluación donde la certeza de clasificación es la medida de desempeño, normalmente se utiliza una aproximación de evaluación basada en particiones de entrenamiento y prueba. Así, la partición de entrenamiento es utilizada para crear el modelo de aprendizaje, mientras que la partición de prueba es utilizada para probar dicho modelo.

5.4.1. Validación cruzada

Validación cruzada tiene sus orígenes en el método hold-out. Hold-out funciona a grandes rasgos de la siguiente forma: toma una fracción de los datos disponibles y los emplea para entrenar a la red, la fracción restante es utilizada para probar la red, esto es, los datos que no son utilizados para entrenar a la red son tomados como si fueran datos en una situación de clasificación real, con la diferencia de que para estos datos se conoce de antemano el valor de la clase; al hacer las pruebas se compara el valor de clase arrojado por el modelo entrenada con el valor de clase real obteniendo así una estimación de error.

Dada la definición anterior puede deducirse que entre más datos se tenga es mejor, el modelo generado se aproxima más al problema real y se tiene la posibilidad de realizar más pruebas, originando resultados más confiables. Desafortunadamente, no siempre se cuenta con muchos datos para entrenar una red. Normalmente el hold-out utiliza dos tercios de los datos para entrenar la red y el tercio restante para probarla. Con pocos datos es más probable que se dé el caso de que los datos no sean representativos. Para evitar este problema, se recomienda que la proporción en los valores de clase sea similar entre los datos de entrenamiento y los de prueba. La estratificación es un procedimiento que se encarga precisamente de separar los datos en porciones significativamente iguales, en términos de valores de la clase.

Una forma de mitigar cualquier prejuicio hecho por las muestras seleccionadas en el método de hold-out es repetir el proceso, entrenando y probando con diferentes muestras de datos escogidas aleatoriamente. En cada iteración una cierta proporción, por ejemplo dos tercios, es escogida aleatoriamente para entrenamiento, posiblemente con estratificación, y la proporción restante para pruebas. La estimación de error en las diferentes iteraciones es promediada para generar una estimación de error global. Este método es conocido como holdout repetido de estimación de razón de error.

Una pequeña variación del método antes descrito forma la base de un método estadístico conocido como validación cruzada. En validación cruzada es posible decidir el

número de particiones (pliegues) que tendrán los datos. Por ejemplo, suponiendo cuatro pliegues, los datos son divididos en cuatro partes aproximadamente iguales, uno de los pliegues es utilizado para pruebas y los tres restantes para entrenamiento, esto es, se utilizan tres cuartos para entrenamiento y un cuarto para pruebas y se repite el procedimiento cuatro veces, de tal modo que cada pliegue sea utilizado una vez para pruebas. El ejemplo antes mencionado se denomina validación cruzada con cuatro pliegues (4-fold cross-validation). Un ejemplo gráfico de este procedimiento puede ser visto en la figura 5.3.

Figura 5.3.: Validación cruzada con 4 pliegues

Usualmente se utiliza validación cruzada estratificada con diez pliegues para hacer la estimación de error de un método de clasificación. Este número de pliegues es el escogido debido a diversas pruebas realizadas que demuestran que este número es adecuado (Kohavi y cols., 1995), aunque no es una medida que deba seguirse forzosamente (Witten y Frank, 2005). El conjunto de datos es dividido aleatoriamente en diez partes en las cuales la clase es representada aproximadamente en la misma proporción que en el conjunto de datos completo. En cada iteración una de las particiones es reservada para pruebas y las nueve restantes son utilizadas para entrenamiento; entonces la razón de error es calculada utilizando la partición reservada para pruebas. El proceso se repite diez veces con diferentes conjuntos de datos para prueba y entrenamiento. Finalmente las diez estimaciones de error son promediadas para producir una estimación de error global. Para un tratamiento formal de la estimación de error antes mencionada referirse a (Kohavi y cols., 1995).

Para obtener una estimación de error más confiable se suele aplicar validación cruzada estratificada con diez pliegues y diez repeticiones, esto quiere decir que se tienen cien iteraciones. Dada la naturaleza aleatoria con que validación cruzada escoge las particiones, al repetir el procesos diez veces se obtienen diferentes conjuntos de datos para pruebas y para entrenamiento, obteniéndose así resultados estadísticamente más significativos.

5.4.2. Prueba T pareada

La comparación entre dos métodos de clasificación puede limitarse solamente a la precisión de clasificación obtenida, pero esta medida aunque útil puede no ser suficien-

te. Para una comparación más exacta se requiere saber si hay diferencias significativas, desde un punto de vista estadístico, entre ambos métodos, de este modo se descarta el hecho de que un método sea mejor por condiciones aleatorias en el proceso de estimación de la precisión de clasificación (Witten y Frank, 2005). Lo que se desea determinar entonces es, qué método de clasificación es mejor o peor en promedio, con respecto a otro.

En este caso la explicación parte del hecho de que se utilizó validación cruzada para obtener la precisión de clasificación y consecuentemente la estimación de error. Cada experimento realizado con validación cruzada da como resultado una estimación de precisión diferente e independiente, lo que interesa es encontrar la media de esas estimaciones para cada método de clasificación y determinar si una media es significativamente mayor o menor que la otra. Lo anterior puede ser logrado mediante el método estadístico prueba t pareada (paired t-test) (Witten y Frank, 2005).

Se cuenta con la siguiente definición (Witten y Frank, 2005): Existe un conjunto de muestras $(x_1, x_2, ...x_k)$ obtenidas mediante validación cruzada con 10 pliegues usando un método de clasificación, y otro conjunto de muestras $(y_1, y_1, ...y_1)$ obtenidas mediante validación cruzada con 10 pliegues usando otro método de clasificación. Denotando a $\bar{x}$ como la media del primer conjunto y a $\bar{y}$ como la media del segundo conjunto. Lo que se trata de determinar es si $\bar{x}$ es significativamente diferente a $\bar{y}$.

Si existen suficientes muestras, la media $\bar{x}$ de un conjunto de muestras independientes $(x_1, x_2, ...x_k)$ tiene una distribución normal (Gaussiana) (Witten y Frank, 2005), a pesar de distribución subyacente de las propias muestras. μ representa el valor verdadero de la media. Si se logra conocer la varianza de esa distribución normal, de tal modo que pueda ser reducida a una media de cero y a una varianza de uno , entonces podrían obtenerse límites de confianza en μ dada la media $\bar{x}$. La varianza de $\bar{x}$ puede ser obtenida dividiendo la varianza obtenida del conjunto $(x_1, x_2, ...x_k)$ entre una variable k , la variable k representa el número de muestras, así mismo la varianza obtenida del conjunto $(x_1, x_2, ...x_k)$ puede ser representada mediante σ_x^2.

La distribución de $\bar{x}$ puede ser reducida para que tenga una media de cero y una varianza de uno mediante: $\frac{\bar{x}-\mu}{\sqrt{\sigma_x^2/k}}$.

Dado que la varianza es sólo una estimación, ésta no tiene una distribución normal(aunque puede ser normal para valores grandes de k). Sin embargo, tiene una distribución conocida como distribución de estudiante con $k-1$ grados de libertad. Lo anterior significa en la práctica que es necesario utilizar una tabla de intervalos de confianza para distribuciones de estudiante. Para nueve grados de libertad los limites de confianza apropiados se muestran en la tabla 5.1. En dicha tabla $Pr[X \geq z]$ representa la probabilidad de que una variable aleatoria X (en este caso X es la razón esperada de éxito) sea mayor o igual al valor de la distribución z. Diferentes tablas son requeridas para diferentes grados de libertad, y si hay más de cien grados de libertad los límites de

confianza se aproximan bastante a los de una distribución normal.

Cuadro 5.1.: Limites de confianza para una distribución de estudiante con nueve grados de libertad

$Pr[X \geq z]$	z
0.1 %	4.30
0.5 %	3.25
1 %	2.82
5 %	1.83
10 %	1.38
20 %	0.88

Para decidir si $\bar{x}$ y $\bar{y}$ (cada uno representa un promedio del mismo número de muestras) son iguales entre sí, se consideran las diferencias d_i entre observaciones que se corresponden, $d_i = x_i - y_i$. La media de esta diferencia es la diferencia entre las dos medias, $d_i = x_i - y_i$, de la misma forma que las medias, $\bar{d}$ tiene una distribución de estudiante con $k - 1$ grados de libertad. Si las medias son iguales, la diferencia es cero (conocida como la hipótesis nula); si son significativamente diferentes, la diferencia así mismo será significativamente diferente a cero. Así que dado un nivel de confianza, se verifica si la diferencia actual excede el límite de confianza. El primer paso es reducir la diferencia a una media de cero y a una unidad de varianza, esta reducción es conocida como estadística-t (t-statistic): $\frac{\bar{d}}{\sqrt{\sigma_x^2/k}}$

Después es necesario decidir el nivel de confianza a utilizar, generalmente 5 % o 1 % es utilizado en la práctica, el límite z puede ser determinado mediante la tabla 5.1 si k es 10; si no lo es, una tabla de confianza de la distribución de estudiante para k es utilizada. Una prueba de dos-colas (two-tailed test) es apropiada porque no se puede conocer por adelantado si la media de las x es mayor a la media de las y o vice versa: así que para una prueba de 1 % se utiliza el valor correspondiente a 0.5 % en la tabla 5.1. Si el valor de t correspondiente a la anterior fórmula es mayor a z , o menor a $-z$, se rechaza la hipótesis nula, que afirma que las medias son iguales, y se concluye que hay una diferencia significativa entre los dos métodos de clasificación en ese dominio para el tamaño del conjunto de datos dado.

Existe un problema con el método anteriormente descrito: sólo indica si un método es preferible a otro, dado un valor determinado de k . Los resultados estimados por validación cruzada no son independientes porque no están basados en conjuntos de datos independientes. En la práctica, esto significa que una diferencia puede ser juzgada como significativa cuando en realidad no lo es. De hecho, al incrementar el número de muestras k , esto es, el número de corridas de validación cruzada, eventualmente puede producirse una diferencia significativa aparente debido a que el valor de la estadística-t se continua

incrementando.

Para lidiar con el problema anteriormente descrito se han producido diversas soluciones que modifican el cálculo estándar de la prueba t (t-test), uno de ellos es la prueba t corregida y remuestreada (corrected resampled t-test) (Witten y Frank, 2005), éste ha probado ser eficaz en la práctica. Este método parte del proceso que hace validación cruzada: cada corrida, $n1$ instancias son usadas para entrenar y $n2$ para probar, y las diferencias d_i son calculadas a partir del rendimiento en los datos de prueba. La fórmula que utiliza este método es la siguiente: $\frac{\bar{d}}{\sqrt{(\frac{1}{k}+\frac{n2}{n1})(\frac{\sigma_x^2}{k}+\frac{\sigma_y^2}{1})}}$

La anterior fórmula no permite que el valor de t se incremente simplemente incrementando k.

Parte II.

Desarrollo

6. Diseño e Implementación

En esta sección se presentan los puntos más importantes relacionados con el diseño y la implementación de la plataforma experimental de aprendizaje así como de la estrategia de aprendizaje colaborativo propuesta que se encuentra embebida en dicha plataforma. Estos puntos son:

- Estrategia de aprendizaje: donde se describe en términos generales la estrategia de aprendizaje propuesta
- Diseño de la plataforma experimental de aprendizaje: donde se presentan las generalidades de la plataforma, los agentes participantes y se describen los artefactos utilizados.
- Flujo de trabajo de un experimento: donde se presenta paso a paso el proceso de aprendizaje bajo la perspectiva de la plataforma antes introducida.
- Capacidades de la plataforma experimental de aprendizaje: donde se especifican las modalidades de uso del sistema, así como los parámetros que pueden configurarse en un experimento en particular.
- Notas sobre eficiencia: en esta sección se mencionan dificultades de eficiencia encontradas y la forma en que se les dio una solución al menos parcial.

6.1. Estrategia de aprendizaje colaborativo

La idea general de la estrategia de aprendizaje colaborativo para ambientes distribuidos propuesta es la siguiente:

- Se tiene una localidad central, en esta localidad se induce un modelo de clasificación base con todos los ejemplos de la localidad. El modelo base puede ser compartido a las demás localidades.
- En cada localidad, secuencialmente, se inicia un proceso mediante el cual se determina si existen contradicciones entre el modelo base y los ejemplos de la localidad. Una contradicción existe si el modelo no predice correctamente la clase del ejemplo.
- Los ejemplos contradictorios son enviados a la localidad central para ser utilizados como datos de entrenamiento en una inducción posterior

- Una vez que una localidad a enviado todos sus ejemplos contradictorios, se determina si se debe realizar una nueva inducción, esto de acuerdo a algún parámetro que lo indique
- Ya sea que se realicé o no una nueva inducción, se pasa a la siguiente localidad para iniciar el proceso de búsqueda de ejemplos contradictorios de nuevo
- El proceso termina cuando ninguna localidad reporta una contradicción, de haberse encontrado alguna, es necesario volver a realizar una revisión en cada localidad dado que el modelo de aprendizaje pudo haber cambiado dando lugar a contradicciones que antes no existían

Esta estrategia general puede adaptarse a los términos de un SMA, teniéndose interacciones colaborativas entre los agentes. En lo que sigue se presenta cómo se realizó esta adaptación, no sólo a los términos de un SMA sino a los de un SMA con ambiente endógeno.

6.2. Diseño de la plataforma experimental de aprendizaje

La plataforma experimental de aprendizaje se refiere al sistema implementado para realizar comparaciones entre la estrategia de aprendizaje colaborativo propuesta y la estrategia de aprendizaje centralizada tradicional. Este sistema utiliza el paradigma de agentes y artefactos, donde los artefactos proveen servicios a los agentes.

Esta plataforma integra la estrategia de aprendizaje discutida en el apartando anterior, adaptándola a términos de agentes y artefactos. Los agentes que componen el sistema son los siguientes:

- Coordinator: existe sólo uno, se encuentra en la localidad central. Se encarga de administrar los ejemplos de la localidad central, coordinar los experimentos, construcción del modelo base, control del proceso de aprendizaje y del manejo de resultados.
- Worker: existe un agente de este tipo en cada localidad (excepto la central), se encarga de administrar los ejemplos de su localidad y de buscar y enviar ejemplos contradictorios de acuerdo a las demandas del coordinador.

Hay tres artefactos principales utilizados por los agentes: `Oracle`, `InstancesBase` y `ClassifierJ48`. El coordinador utiliza el artefacto `Oracle` para extraer meta-información del conjunto de datos que se procesará y para dividir y repartir el conjunto de datos entre los diversos agentes. Cada agente almacena sus ejemplos de entrenamiento en un artefacto de tipo `InstancesBase`. El coordinador induce el modelo base con sus ejemplos utilizando el artefacto `ClassifierJ48`, después pregunta por ejemplos contradictorios a los agentes trabajadores. En la figura 6.1 se presenta una vista general de éstos tres artefactos principales y los agentes. Una descripción más detallada de estos artefactos y del resto de ellos se presenta a continuación.

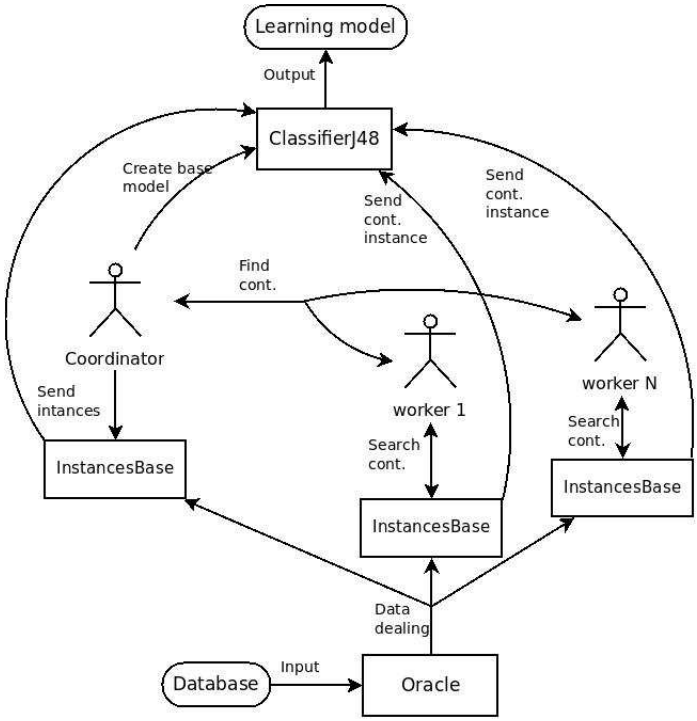

Figura 6.1.: Vista general del sistema

6.2.1. Artefactos

Los artefactos encapsulan operaciones de WEKA, para esto se utilizó WEKA como biblioteca externa haciendo uso del clasificador J48 (C4.5), así como de las herramientas para tratar con archivos arff (para más información de los elementos de WEKA que se utilizaron en el presente ver sección 6.4). Los artefactos CArtAgO que conforman el ambiente son los siguientes:

- *Oracle*: Dado que uno de los objetivos de la plataforma aquí discutida es crear experimentos de aprendizaje distribuidos, es necesario particionar conjuntos de datos existentes de una forma controlada para ser capaces de realizar comparaciones. El artefacto Oracle, crea particiones estratificadas aleatorias de los datos, éstas son comunicadas a los agentes interesados, así mismo, comunica las características del conjunto de datos (nombre y tipo de los atributos, etc.), de este modo los agentes y el artefacto de clasificación son independientes del artefacto Oracle. El agente guarda estas características del conjunto de datos como una creencia, a partir de esta creencia el agente puede comunicar al artefacto J48 los datos relevantes de la base de datos para que este artefacto sea capaz de crear modelos de aprendizaje.

  Las operaciones que este artefacto ofrece a los agentes son las siguientes:

  - dataSetInfo: devuelve los metadatos de la base de datos con que trabaja Oracle. Estos metadatos son almacenados como una creencia por el agente, pudiendo ser pasados al artefacto ClassifierJ48 por intermediación del agente
  - numTotalInstances: regresa el número de instancias totales que contiene la base de datos. Utilizada en operaciones estadísticas
  - restartOracle: reinicializa las particiones de datos. Es útil para manejar las diversas repeticiones que un experimento puede tener

- recreateOracle: recrea el esquema de particiones para el caso de repeticiones de Validación cruzada
- reInitOracle: crea nuevamente todo el artefacto. Esta operación se utiliza cuando se quiere correr un experimento nuevo que puede tener otros parámetros

Las operaciones que este artefacto ofrece a otro artefacto son las siguientes:

- givePartition: entrega la partición de datos correspondiente al artefacto InstancesBase
- getTesting: entrega todos los datos de entrenamiento. Esta operación es invocada por Evaluator
- getTraining: entrega todos los datos de entrenamiento. Esta operación es utilizada para el experimento centralizado tradicional que utiliza todos los datos de entrenamiento

- *ClassifierJ48*: es el artefacto principal de aprendizaje, implementa el algoritmo C4.5, pudiéndose activar o desactivar el pruning. El artefacto se encarga de recibir ejemplos, inducir árboles de decisión a partir de los ejemplos, compartir su modelo aprendido y clasificar ejemplos (aunque la operación de clasificación no se utiliza directamente en el presente). Cabe decir que este artefacto está pensado para ser independiente, tanto de otros artefactos (aunque por razones experimentales tiene una operación que depende del artefacto Oracle) así como de la base de datos (aunque requiere archivos arff) por lo que esta Implementación puede ser reutilizada. La independencia de la base de datos se consigue gracias al hecho de que los metadatos concernientes a la base de datos (nombre, tipo y rangos de valores de los atributos) son recibidos por este artefacto como argumento. Así el agente coordinador tiene una creencia concerniente a los metadatos, la cual obtiene gracias al artefacto Oracle, estos metadatos son pasados en la inicialización del artefacto ClassifierJ48.

  Las operaciones que este artefacto ofrece a los agentes son las siguientes:

  - buildClassifier: lleva a cabo la inducción del modelo
  - classifyInstance: regresa la clase de una instancia recibida. Esta operación existe por completitud ya que en el presente no se utiliza, las clasificaciones requeridas son en realidad llevadas a cabo por el artefacto InstancesBase por razones de eficiencia (ver sección 6.5)
  - getTrainDataFromOraculo: provoca que el artefacto Oracle envíe todos los ejemplos de entrenamiento a este artefacto. Esta operación es útil en la evaluación centralizada tradicional donde todos los ejemplos de entrenamiento son utilizados
  - numTrainInstances: Regresa el número de instancias de entrenamiento que este artefacto posee. Esta operación es útil para realizar conteos estadísticos

- printTree: imprime en consola el árbol de decisión actual
- reInitJ48: reinicializa el artefacto. Útil para el esquema experimental, donde se necesitan llevar a cabo varias corridas
- restartJ48: reinicializa los ejemplos de entrenamiento. Útil en el caso de validación cruzada donde las particiones de entrenamiento y prueba rotan

Las operaciones que este artefacto ofrece a otro artefacto son las siguientes:

- getModel: envía el modelo actual al artefacto que lo solicita. Utilizado por InstancesBase para realizar clasificaciones de forma más eficiente
- addInstance: utilizado por InstancesBase, añade una instancia de entrenamiento al artefacto

- *InstancesBase*: cada agente tiene relacionado un único artefacto de este tipo. El artefacto se encarga de almacenar y gestionar los ejemplos del agente. Este artefacto se encuentra ligado al artefacto ClassifierJ48 y Oracle para poder realizar operaciones de forma más eficiente.

  Las operaciones que este artefacto ofrece a los agentes son las siguientes:

  - feedExamples: realiza una solicitud al artefacto Oracle para que éste envíe los ejemplos de entrenamiento correspondientes.
  - sendAllExamples: envía todas las instancias al artefacto ClassifierJ48. Esta operación es útil para el coordinador que debe enviar todos sus ejemplos de entrenamiento para crear el modelo base
  - searchSendContradictions: obtiene el modelo de entrenamiento actual, mediante este se realiza una búsqueda de contradicciones en la base de ejemplos, de encontrarse una contradicción, el ejemplo es enviado al artefacto ClassifierJ48

- GUI: artefacto que representa un front-end para realizar experimentos, puede especificarse el conjunto de datos, el número de repeticiones del experimento, el método de prueba a usarse (validación cruzada o hold-out), el número de agentes participantes, etc. (para una lista completa de los parámetros y su significado ver sección 6.3). Estos parámetros son percibidos por el agente y se almacenan como creencias, el agente coordinador utiliza estas creencias para configurar los diversos artefactos y el experimento en general.

  Las operaciones que este artefacto ofrece a los agentes son las siguientes:

  - printResult: imprime un resultado en el área de resultados de la ventana. Esta operación es utilizada varias veces para imprimir los diversos resultados generados

- activateStart: reactiva el botón de start que es desactivado por defecto mientras un experimento se encuentra en curso. Operación utilizada cuando un experimento ha concluido

- Evaluator: Se encargar de realizar operaciones estadísticas con los resultados arrojados por el experimento. Estas operaciones comprenden medias, desviación estándar y prueba T pareada.

  Las operaciones que este artefacto ofrece a los agentes son las siguientes:

  - getModelFromClassifierJ48: recupera el modelo inducido por ClassifierJ48 para realizar diversas evaluaciones
  - getTestDataFromOraculo: recupera los ejemplos de prueba del artefacto Oracle
  - evaluate: lleva a cabo una evaluación de certeza de clasificación
  - pairedTTest: recibe dos conjuntos de resultados de certeza de clasificación y determina si existen difetencias significativas entre ambos resultados
  - mean: devuelve la media de un conjunto de resultados de certeza de clasificación
  - stdDev: devuelve la desviación estándar de un conjunto de resultados

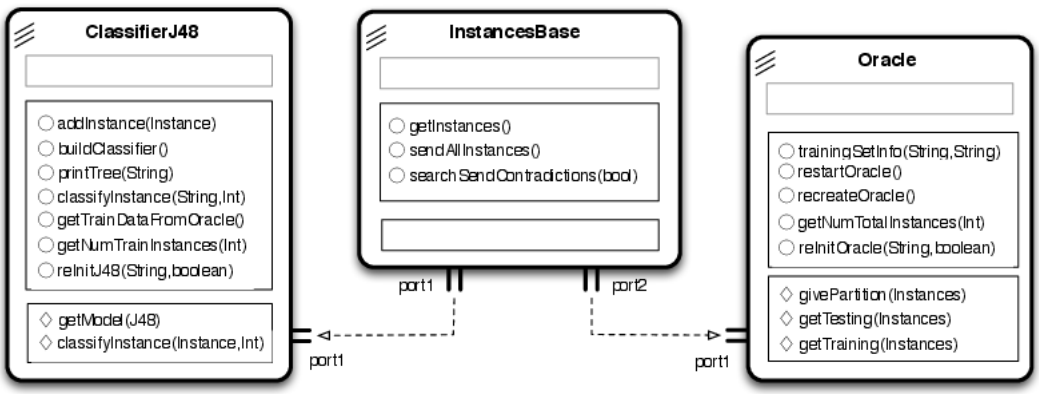

Figura 6.2.: Interacción de los artefactos pricipales

6.2.2. Flujo de trabajo de un experimento

Antes de que un experimento comience, los parámetros del experimento son configurados a través del artefacto GUI, estos parámetros son detallados en la sección 6.3. Un experimento tiene el siguiente flujo general de trabajo:

- El coordinador determina que agentes van a participar en el experimento. Actualmente todos los agentes participan, sin embargo, se deja abierta la posibilidad para que algunos agentes no lo hagan. Esta decisión beneficia la flexibilidad del sistema
- El coordinador crea y configura los artefactos necesarios
- El coordinador le solicita al artefacto Oracle su partición de datos
- Cada trabajador le solicita a Oracle su partición de datos
- El coordinador le envía todos sus datos de entrenamiento a ClassifierJ48
- ClassifierJ48 induce un árbol con estos datos para crear el modelo base. Recordando, el modelo base es el primer modelo que se genera presumiblemente con pocos ejemplos, y como su nombre lo indica, sirve como punto de partida para iniciar el proceso de aprendizaje colaborativo
- El coordinador inicia el proceso colaborativo, preguntando a cada trabajador, uno por uno, si tiene ejemplos contradictorios
- Si se encuentran contradicciones, se le pide a ClassifierJ48 que añada los ajemplos y realice otra inducción posteriormente (el momento de la inducción puede variar de acuerdo a un parámetro, ver sección 6.3). Así mismo, los ejemplos añadido son borrados de la base de ejemplos del agente. De esta forma el proceso de aprendizaje siempre termina, evitándose así ciclos infinitos
- Si se ha hecho una nueva inducción, es necesario que todos los trabajadores vuelvan a realizar la búsqueda de ejemplos contradictorios, volviendo a comenzar de alguna forma todo el proceso, excepto que el modelo va mejorando en cada iteración. Es necesario volver a iniciar el proceso ya que al cambiar el modelo es posible que se encuentren nuevas contradicciones que antes no habían sido detectadas.
- El proceso continua hasta que ya no se encuentren contradicciones en los ejemplos o hasta que los ejemplos se agoten
- Si hay varias repeticiones del experimento, éstas repeticiones se realizan, las repeticiones sirven para dar soporte estadístico a las pruebas. Luego, el agente coordinador reúne los datos estadísticos y los imprime a través del artefacto GUI

Los siguientes diagramas de interacción resumen las partes más importantes del flujo de trabajo antes descrito. En la figura 6.3 se muestra el proceso que se lleva a cabo para repartir los datos, mientras que en la figura 6.4 se muestra el proceso de aprendizaje.

En estos diagramas se obvia la presencia de los artefactos InstancesBase para que los diagramas sean más fáciles de leer.

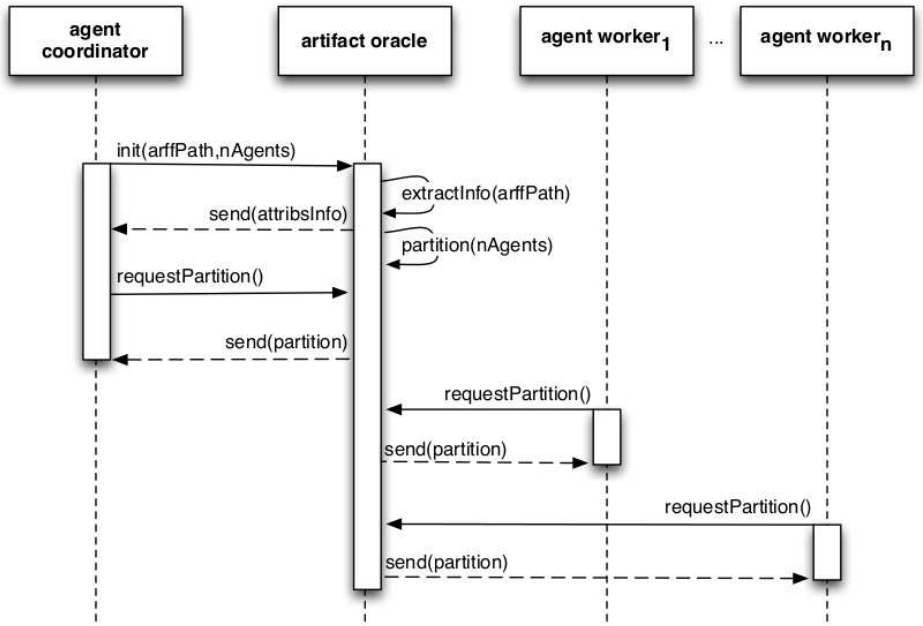

Figura 6.3.: Diagrama de interacción, distribución de datos

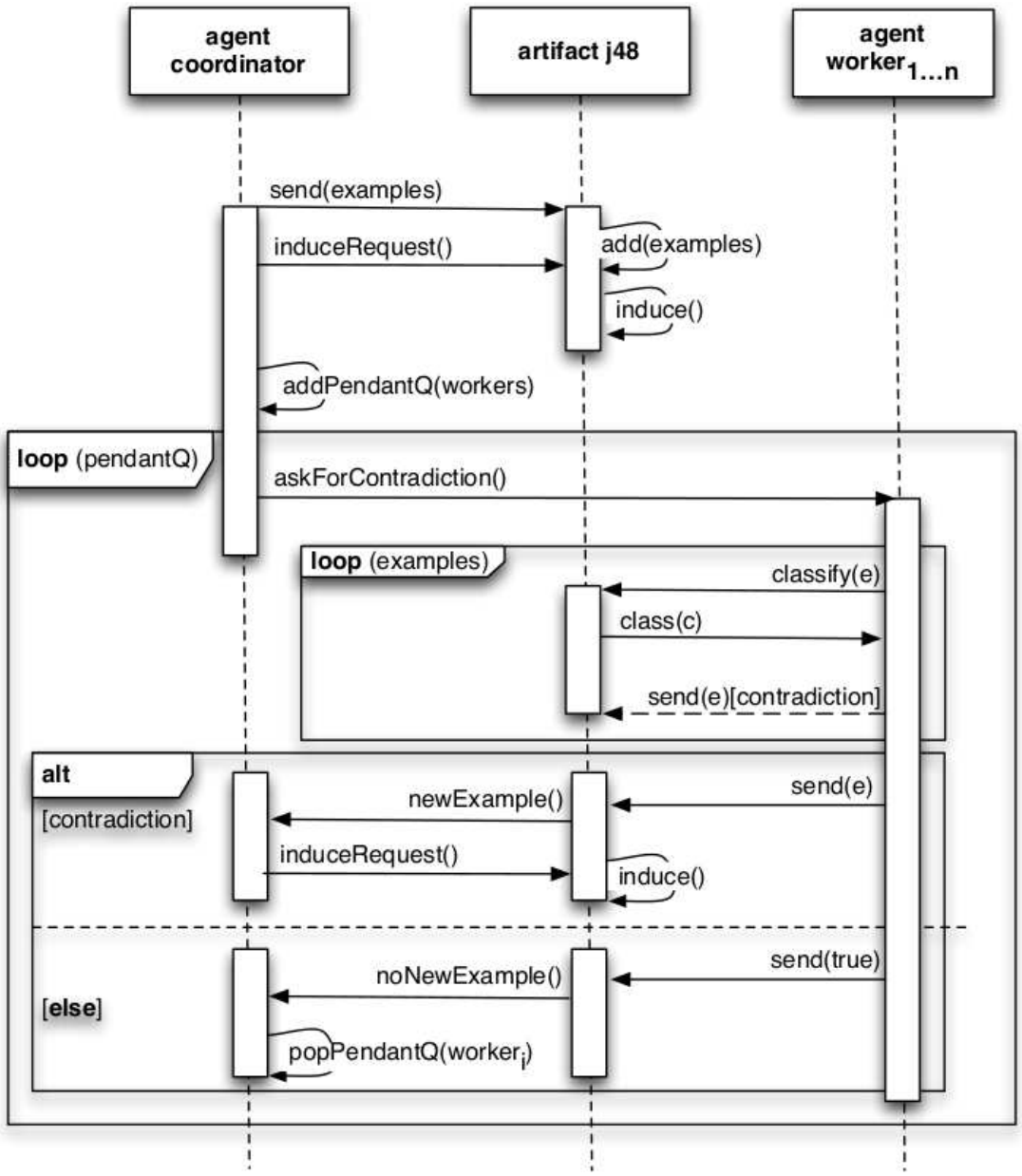

Figura 6.4.: Diagrama de interacción, proceso de aprendizaje

6.3. Capacidades de la plataforma experimental de aprendizaje

El sistema introducido en secciones anteriores cuenta con dos modalidades de uso: modo GUI y modo experimental. El modo GUI es utilizado para realizar experimentos individuales, mientras que en el modo experimental pueden configurarse varios experimentos mediante un script Python. Los parámetros con los cuales puede configurarse un experimento en cualquiera de las dos modalidades son los siguientes:

- Archivo o archivos (en el caso de modo experimental) arff a utilizar: los archivos deben ser archivos arff de clasificación válidos. El sistema puede tratar con archivos que contienen datos incompletos.
- Número de repeticiones del experimento: este parámetro da la posibilidad de tener resultados estadísticamente más significativos. Recordando que para validación cruzada el número de pliegues también determina un número de repeticiones. Así de tenerse un experimento con validación cruzada con 10 pliegues y 10 repeticiones, lo que se tiene es un experimento que producirá 100 modelos de aprendizaje, 10 modelos en cada repetición
- Número de agentes trabajadores: al menos debe haber uno. Dependiendo del número de agentes la cantidad de ejemplos repartido por agente varía
- Tipo de prueba: Hold-out o validación cruzada
- En el caso de Hold-out, porcentaje de datos utilizado para entrenamiento
- En el caso de validación cruzada, número de pliegues utilizados
- Número de revisiones máximas a las cuales un ejemplo puede ser sujeto (ver sección 6.5)
- Porcentaje de agentes revisados antes de inducir (ver sección 6.5)
- Incremento del porcentaje de agentes revisados antes de inducir (ver sección 6.5)

En el caso de la modalidad GUI, los resultados de los experimentos son presentados en la ventana principal, mientras que en el modo experimental se arroja un archivo LaTeX con tablas de resultados. En la figura 6.5 se muestra el sistema corriendo tras haber ejecutado un experimento.

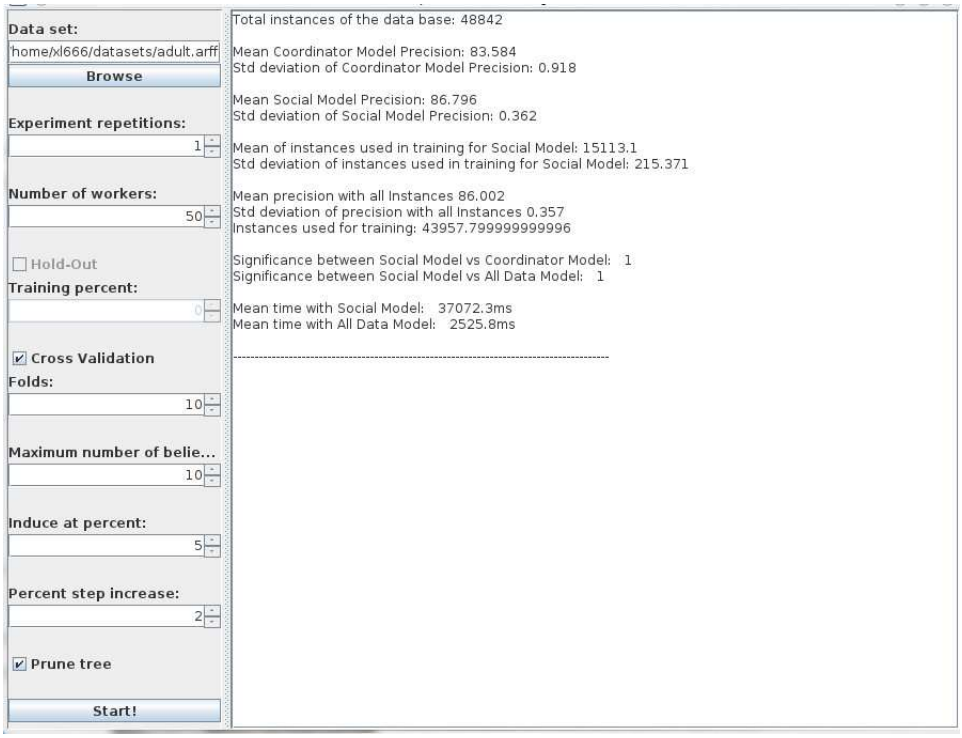

Figura 6.5.: Interfaz gráfica del sistema

6.4. WEKA

WEKA (Hall y cols., 2009), por sus siglas: Waikato Environment for Knowledge Analysis, es una aplicación implementada en java protegida por la licencia GPL, que reúne varios métodos de aprendizaje automático y minería de datos con el fin de integrarlos en un ambiente común. Así mismo, WEKA cuenta con una plataforma experimental que permite realizar comparaciones entre métodos.

WEKA integra métodos de carga y pre-procesamiento de datos, algoritmos para crear modelos de clasificación, algoritmos de clustering, algoritmos de regresión y métodos de evaluación de modelos. WEKA hace accesibles todos estos algoritmos y métodos a través de su API. De este modo es posible utilizar WEKA como biblioteca externa. En el presente se utilizó WEKA como biblioteca externa para cumplir con las siguientes tareas:

- Carga de archivos arff: mediante clase Instances. Instances es una colección de ejemplos
- Tratar datos incompletos: mediante la clase ReplaceMissingValues, que remplaza valores por la media o moda según sea el caso

- Evaluación de certeza de clasificación: mediante la clase Evaluation. Evaluation sólo toma una base de datos de entrenamiento y otra de prueba, el proceso de validación cruzada y holdout es llevado a cabo de forma manual
- Realización de pruebas de hipótesis: mediante la clase PairedStats. Esta clase recibe dos arreglos de dobles que contienen resultados de certeza de clasificación de dos métodos diferentes. Arroja un entero que determina si hay o no diferencias a favor en en contra del primer método
- Inducción de árbol de decisión: mediante la clase J48. J48 contiene el algoritmo de C4.5. WEKA permite dar una serie de opciones a los clasificadores donde es posible determinar los parámetros relevantes para el clasificador. En el caso de J48 se dejaron las opciones por defecto y es posible activar o desactivar el pruning mediante una opción del GUI o el script de experimentación
- Clasificar instancias: todos los clasificadores de WEKA tienen una interfaz en común que integra la operación de clasificación

6.5. Notas sobre eficiencia

A lo largo del desarrollo de este trabajo, se realizaron diversas mejoras de eficiencia tanto de la estrategia de aprendizaje como del sistema que la implementa. Estas mejoras fueron hechas con el afán de ser capaces de probar bases de datos grandes (más de 3000 ejemplos) en un tiempo razonable, y no tanto con la intención de tener un sistema lo más eficiente posible ya que por el momento esa no es la intención de este trabajo. Sin embargo, la necesidad de realizar estas mejoras permitió llevar a cabo un análisis de los puntos más importantes para mejorar la eficiencia, algunos de ellos son abordados a continuación.

La optimización general más importante que se realizó fue mover los ejemplos de entrenamiento a un artefacto a parte (artefacto InstancesBase). En una versión anterior, los ejemplos de entrenamiento eran almacenados en la base de creencias de los agentes. Cuando se requería hacer la búsqueda de ejemplos contradictorios, cada ejemplo era enviado desde el agente hasta el artefacto ClassifierJ48, ya que el agente hace una solicitud de clasificación a este artefacto. Estas solicitudes de clasificación son muy frecuentes durante la búsqueda de ejemplos contradictorios y era en este punto donde existía un cuello de botella. Los agentes están a un nivel AgentSpeak, mientras que los artefactos están a un nivel Java. En cada solicitud era necesario realizar el cambio de nivel, sin mencionar que el borrar, buscar y mover ejemplos desde la base de conocimientos del agente (nivel AgentSpeak) es ineficiente cuando se compara con la manipulación de objetos Java. En la implementación actual, la comunicación entre agentes y los demás artefactos, así como la administración general de ejemplos, se lleva a cabo en un artefacto a parte (cada agente tiene asociado un artefacto) y por lo tanto a un nivel Java, la comunicación entre artefactos es mucho más rápida que la comunicación entre agente y artefacto. Con este

cambio la eficiencia del sistema mejoró considerablemente, teniéndose ahora la posibilidad de probar bases de datos con muchos ejemplos.

En cuanto a mejoras de rendimiento más especificas, en este esquema se han identificado tres puntos clave que afectan directamente a la eficiencia, estos puntos son los siguientes:

- Envío de instancias
- Revisiones de instancias
- Inducciones

Para cada uno de estos puntos se llevó a cabo una optimización. A continuación se elabora cada uno de estos puntos.

6.5.1. Envío de instancias

El envío de instancias se refiere a la transmisión de datos. Existen envíos que son inevitables, estos son: el envío de ejemplos inicial desde el oráculo a los agentes, el envío de instancias del coordinador al artefacto ClassifierJ48 para crear el modelo base y el envío de ejemplos contradictorios de los agentes trabajadores al artefacto ClassifierJ48. Estos envíos son inevitables debido a que o bien forman parte de la configuración experimental (en el caso del envío inicial) o bien son parte fundamental del esquema actual (recordando que un futuro existe la posibilidad de modificar el esquema).

Existe otro envío que puede ser constante, este envío es el que hacen los agentes trabajadores al artefacto ClassifierJ48, estos envíos son peticiones de clasificación, en estas peticiones la instancia viene incluida en el mensaje. El proceso de búsqueda de ejemplos contradictorios es uno de los más costosos del sistema, esto debido a que hacen falta varias revisiones de los ejemplos para llegar a un punto sin contradicciones, estas revisiones incluyen una clasificación, teniéndose un costo de transmisión directamente proporcional al número de ejemplos. En versiones anteriores del sistema, se tenía precisamente esta forma de trabajo, cada vez que hacia falta clasificar una instancia se le hacía una petición al artefacto ClassifierJ48. En la versión que se presenta se ha mejorado este aspecto. En este esquema, antes de iniciar la búsqueda de ejemplos contradictorios, el agente trabajador le solicita al artefacto ClassifierJ48 su modelo de aprendizaje, este modelo es un objeto que tiene la operación de clasificación integrada. Entonces, en vez de transmitir muchas instancias lo que se transmite es un modelo, no sólo se ahorra tiempo en la transmisión de la instancia, sino que también en el tiempo de respuesta que tiene la petición del agente al artefacto (aunque en realidad se hace por inmediación del artefacto que maneja las instancias del agente) ya que la clasificación se hace de forma interna.

6.5.2. Revisiones de instancias

Una optimización más fue realizada. Esta optimización consiste en limitar el número de veces que un mismo ejemplo es considerado en la búsqueda de ejemplos contradictorios. Si un ejemplo es considerado consistente, éste es devuelto a la base de ejemplos, si fuera borrado directamente, lo que indudablemente aumenta la eficiencia del sistema, se estaría cometiendo un error, puesto que nunca se sabe cuando un ejemplo puede volverse contradictorio dado que el modelo de aprendizaje cambia constantemente. Sin embargo, conservar todos los ejemplos consistentes plantea re-revisar muchas veces el mismo ejemplo, lo cual es costoso ya que implica clasificarlo de nuevo. En la implementación actual es posible limitar el número de veces que un ejemplo es revisado, esto se logra mediante un parámetro.

Esta limitación en el número de revisiones también limita de alguna forma el número de revisiones totales que hace el sistema y dependiendo de su valor puede mejorar enormemente la eficiencia del mismo. Sin embargo, este parámetro es un arma de doble filo, si se escoge un valor muy pequeño es posible que la certeza de clasificación de los modelos aprendidos disminuya considerablemente, en la mayoría de bases de datos probadas este valor debe ser al menos 10 para mantener una buena precisión, lo que indica que el proceso de hacer múltiples revisiones a una sola instancia tiene sentido.

6.5.3. Inducciones

La inducción de modelos de aprendizaje se vuelve más costosa conforme el número de instancias aumenta. Para bases de datos pequeñas este costo de inducción es despreciable, pero entre más instancias se tienen se comienza a ver que este proceso es posiblemente el más costoso. En un principio, se tenía un esquema simple, donde la búsqueda de ejemplos contradictorios se hacia uno a la vez, induciendo un nuevo modelo cada vez que se encontraba, en lugar del actual donde cada agente busca todos sus ejemplos contradictorios y una vez que los tiene se lleva a cabo la inducción (aunque en realidad la inducción depende también de otros parámetros que se mencionan más adelante). Cabe decir que con el esquema anterior se realizaban muchas más inducciones, lo cual bajaba considerablemente el rendimiento del sistema. Con la optimización realizada el número de inducciones disminuyó considerablemente, mientras que no se observaron muchas diferencias en cuanto el número de ejemplos de entrenamiento totales utilizados.

Además de la mejora antes mencionada, existen dos parámetros con los que es posible controlar, hasta cierto grado, la frecuencia con que se realizan inducciones. Con el primero de estos parámetros es posible indicar que porcentaje de los agentes necesitan ser revisados antes de que se induzca, por ejemplo, si hay 100 agentes y en este parámetro se indica 20, eso quiere decir que se debe inducir una vez que hallan pasado 20 agentes, la siguiente inducción se hará cuando hayan pasado 40 y así sucesivamente. El segundo parámetro sirve para incrementar el parámetro antes mencionado conforme el proceso avanza. Este segundo parámetro hace que el porcentaje de inducción inicial se incremen-

te después de cada ciclo de evaluación. Así, por ejemplo, si el valor de ese parámetro es 5, eso quiere decir que al siguiente ciclo se evaluará al 25 % de agentes (si tomamos en cuenta que el valor del primer parámetro es 20). Eventualmente el porcentaje puede llegar a ser 100 (esto depende también del número de revisiones). Si no se quiere hacer uso de estos parámetros (para tener un comportamiento igual a la de versiones anteriores) simplemente se dejan con valor 0. Estos parámetros tienen un impacto en el número de ejemplos de entrenamiento, si no se escogen bien es posible que se utilicen más ejemplos de entrenamiento, aunque con un tiempo más reducido.

7. Metodología experimental

Se lleva a cabo una comparación entre el esquema propuesto y la estrategia centralizada tradicional en los siguientes aspectos:

- Número de instancias utilizadas para entrenar
- Certeza de clasificación
- Tiempo

A pesar de que el sistema experimental de aprendizaje implementado es conceptualmente distribuido, los experimentos fueron realizados en una única computadora, simulándose la existencia de diversas localidades. Esto fue de esta forma debido a que el interés principal del presente es verificar el número de ejemplos de entrenamiento utilizados para inducir el modelo de aprendizaje así como la certeza de dicho modelo. Para cumplir con este objetivo no era necesario realizar pruebas en un sistema distribuido real, y desde un punto de vista de configuración y recursos experimentales, resulta más sencillo realizar los experimentos en una sola computadora. Sin embargo, es obvio como esta decisión afecta los resultados de tiempo presentados. Dichos resultados podrían no ser del todo realistas ya que no se toma en consideración el tiempo de transmisión de los ejemplos de una localidad a otra.

Los experimentos fueron realizados utilizando algunas de las bases de datos del repositorio UCI (Bache y Lichman, 2013) (ver tabla 7.1), se utilizó un esquema de particionamiento estratificado aleatorio para repartir los datos entre los diversos agentes, la estratificación permite que todas las particiones de datos mantengan una proporción similar de los valores de clase, el artefacto Oracle es el encargado de esta operación. Las pruebas de certeza de clasificación fueron realizadas utilizando validación cruzada estratificada con 10 pliegues y 10 repeticiones (osea que se produjeron 100 modelos), con excepción de la base de datos poker la cual por su tamaño sólo se probó con una repetición. Se realizaron pruebas utilizando 1, 10, 30, 50 y 100 agentes trabajadores. El método propuesto fue comparado contra una versión centralizada tradicional, donde todos los datos de entrenamiento son utilizados para construir el modelo. Para realizar las comparaciones entre métodos se utilizaron las mismas particiones de datos. Las comparaciones son en cuanto a certeza de clasificación, número de ejemplos de entrenamiento utilizados y tiempo total en milisegundos que utilizó cada método para llegar a su modelo final, se utilizan milisegundos ya que para las bases de datos pequeñas es más fácil hacer una comparación. Se presentan resultados de prueba T pareada de dos colas con 0.05 grados de significancia para determinar si en las comparaciones hay diferencias significativas en

cuanto a la certeza de clasificación. Notar que no se utilizó la versión corregida de la prueba T pareada de dos colas, que usualmente se emplea en validación cruzada (Hall y cols., 2009)(y que es menos propensa a señalar diferencias significativas) debido a que la versión corregida requiere de un único número de instancias de entrenamiento como parámetro de cálculo. Como puede esperarse, el número de instancias de entrenamiento no es el mismo para el experimento centralizado que para el colaborativo, por lo tanto no puede aplicarse directamente el método corregido. Se realizaron además experimentos con y sin pruning activo.

En los experimentos pueden establecerse, además del número de repeticiones y pliegues para validación cruzada, los siguientes parámetros:

- Número de revisiones: este parámetro determina cuantas veces un mismo ejemplo puede ser evaluado en la búsqueda de ejemplos contradictorios. Entre más pequeño sea el valor de este parámetro, más rápido sera el proceso (se requerirán menos revisiones), sin embargo, de no escogerse bien este valor se corre el riesgo de perder certeza de clasificación.

- Porcentaje de agentes revisados antes de inducir: este parámetro determina el porcentaje de agentes que deben haber sido revisados en la búsqueda de ejemplos contradictorios antes de realizar una inducción. Entre más grande sea el valor de este parámetro (tendiéndose un valor máximo de 100) más rápido será el proceso (se requieren menos inducciones), sin embargo, de no escogerse bien el valor de este parámetro es posible que se utilicen muchos más ejemplos de entrenamiento de los necesarios.

- Incremento del porcentaje de agentes revisados antes de inducir: determina el incremento en el porcentaje

Para más información sobre los parámetros antes mencionados ver la sección 6.5. La tabla 7.2 muestra los valores escogidos para los parámetros antes mencionados dependiendo de la base de datos utilizada y el número de agentes. Estos valores fueron escogidos con base a observaciones empíricas y no se descarta la existencia de otros valores que arrojen mejores resultados.

Cuadro 7.1.: Bases de datos probadas

Base de datos	#Registros	#Atributos	#Clases
adult	48842	15	2
australian	690	15	2
breast	683	10	2
german	1000	21	2
heart	279	14	2
iris	150	5	3
kr-vs-kp	3196	37	2
letter	20000	17	26
poker	829201	11	10
waveform-5000	5000	41	3

Cuadro 7.2.: Parámetros experimentales escogidos dependiendo de la base de datos y número de agentes

Base de datos	#Agentes	#Revisiones	% Inducción	Inc % Inducción
adult	1	5	0	0
adult	10	5	10	1
adult	30	5	10	2
adult	50	5	15	3
adult	100	5	20	5
australian	1-100	Ilimitadas	0	0
breast	1-100	Ilimitadas	0	0
german	1	10	0	0
german	10	10	10	1
german	30	10	10	2
german	50	10	15	3
german	100	10	20	5
heart	1-100	Ilimitadas	0	0
iris	1-100	Ilimitadas	0	0
kr-vs-kp	1-100	Ilimitadas	0	0
letter	1	10	0	0
letter	10	10	10	1
letter	30	10	10	2
letter	50	10	15	3
letter	100	10	20	5
poker	1	2	0	0
poker	10	2	10	1
poker	30	2	10	2
poker	50	2	15	3
poker	100	2	25	5
waveform-5000	1	10	0	0
waveform-5000	10	10	10	5
waveform-5000	30	10	10	5
waveform-5000	50	10	30	5
waveform-5000	100	10	30	5

8. Resultados

A continuación se presentan los resultados experimentales arrojados. Estos resultados están divididos en dos partes, una parte se corresponde a los experimentos sin utilizar pruning y la otra a los experimentos utilizando pruning. A su vez, cada una de las partes se divide en tres, presentándose resultados para número de instancias de entrenamiento utilizadas, certeza de clasificación y tiempo.

En el cuadro 8.1 y 8.4 se presenta el número de ejemplos utilizados por la estrategia centralizada, el modelo base y el modelo social. En el caso del modelo social, la desviación estándar es mostrada, esto es debido a que existen variaciones en cada experimento (la desviación estándar es de 100 experimentos).

Los cuadros 8.2 y 8.5 muestran los resultados de certeza de clasificación para las diversas estrategias, así como los resultados de prueba T pareada. Los resultados de prueba T pareada se presentan en la forma de enfrentamientos entre dos modelos emparejados, así, si el resultado es 1, quiere decir que el primer modelo emparejado es significativamente mejor al segundo; si es 0, significa que no existen diferencias significativas entre ambos modelos; finalmente, si es -1 significa que el primer modelo emparejado es significativamente peor al segundo.

Finalmente los cuadros 8.3 y 8.6 presenta el tiempo medio de creación del modelo final para la estrategia centralizada y la estrategia social. El tiempo de la estrategia social incluye el tiempo de creación del modelo base. No olvidar que los tiempos presentados se encuentran en milisegundos.

Cuadro 8.1.: Comparación de instancias de entrenamiento sin utilizar pruning

BD	**#AgTrab**	**#Ej BD**	**#Ej Entr Cen**	**#Ej Entr Base**	**#Ej Entr Colab**
adult	1	48842	43957	21978	28577.33 ± 133.05
adult	10	48842	43957	3996	18379.56 ± 141.57
adult	30	48842	43957	1417	17634.59 ± 151.3
adult	50	48842	43957	861	17924.52 ± 163.77
adult	100	48842	43957	435	18616.15 ± 169.72
australian	1	690	621	310	397.76 ± 9.47
australian	10	690	621	56	250.68 ± 12.41
australian	30	690	621	20	236.06 ± 10.66

Continua en la siguiente página

Cuadro 8.1 – continuación de la página previa

BD	#AgTrab	#Ej BD	#Ej Entr Cen	#Ej Entr Base	#Ej Entr Colab
australian	50	690	621	12	233.54 ± 10.82
australian	100	690	621	6	229.7 ± 10.85
breast	1	683	614	307	341.51 ± 8.15
breast	10	683	614	55	141.45 ± 9.53
breast	30	683	614	19	117.88 ± 9.1
breast	50	683	614	12	111.1 ± 10.11
breast	100	683	614	6	109.51 ± 9.69
german	1	1000	900	450	727.94 ± 11.81
german	10	1000	900	81	649.52 ± 16.48
german	30	1000	900	29	645.59 ± 15.67
german	50	1000	900	17	653.71 ± 17.56
german	100	1000	900	8	662.6 ± 16.12
heart	1	270	243	121	180.2 ± 6.73
heart	10	270	243	22	142.64 ± 7.74
heart	30	270	243	7	137.6 ± 8.0
heart	50	270	243	4	135.83 ± 8.6
heart	100	270	243	2	137.07 ± 8.85
iris	1	150	135	67	72.93 ± 2.64
iris	10	150	135	12	29.62 ± 4.11
iris	30	150	135	4	26.92 ± 3.68
iris	50	150	135	2	26.18 ± 3.88
iris	100	150	135	1	25.73 ± 3.7
kr-vs-kp	1	3196	2876	1438	1464.59 ± 8.84
kr-vs-kp	10	3196	2876	261	348.61 ± 13.22
kr-vs-kp	30	3196	2876	92	207.28 ± 10.81
kr-vs-kp	50	3196	2876	56	181.73 ± 11.51
kr-vs-kp	100	3196	2876	28	164.62 ± 10.73
letter	1	20000	18000	9000	11829.96 ± 158.9
letter	10	20000	18000	1636	8285.92 ± 263.27
letter	30	20000	18000	580	8273.14 ± 213.44
letter	50	20000	18000	352	8356.66 ± 249.33
letter	100	20000	18000	178	8612.84 ± 258.08
poker	1	829201	746280	373140	374903.9 ± 115.74
poker	10	829201	746280	67843.71	76571.1 ± 773.10
poker	30	829201	746280	24073.57	45557.3 ± 992.97
poker	50	829201	746280	14632.95	53271.2 ± 1400.01
poker	100	829201	746280	7388.91	85626 ± 2835.55
waveform	1	5000	4500	2250	3796.32 ± 28.83
waveform	10	5000	4500	409	3541.6 ± 38.62

Continua en la siguiente página

Cuadro 8.1 – continuación de la página previa

BD	#AgTrab	#Ej BD	#Ej Entr Cen	#Ej Entr Base	#Ej Entr Colab
waveform	30	5000	4500	145	3526.2 ± 36.33
waveform	50	5000	4500	88	3553.92 ± 36.16
waveform	100	5000	4500	44	3582.09 ± 52.86

Cuadro 8.2.: Comparación de certeza de clasificación sin utilizar pruning

BD	#AgTrab	Certeza Cen	Certeza Base	Certeza Colab	ColvsCen	ColvsBas
adult	1	84.31 ± 0.47	83.72 ± 0.56	84.67 ± 0.47	1	1
adult	10	84.24 ± 0.53	82.31 ± 0.67	84.63 ± 0.49	1	1
adult	30	84.28 ± 0.46	81.36 ± 0.75	84.63 ± 0.49	1	1
adult	50	84.28 ± 0.43	80.75 ± 0.94	84.69 ± 0.53	1	1
adult	100	84.28 ± 0.51	79.71 ± 1.36	84.67 ± 0.6	1	1
australian	1	83.83 ± 4.43	84.0 ± 4.79	83.71 ± 4.44	0	0
australian	10	83.5 ± 4.33	82.4 ± 5.15	83.95 ± 4.5	0	1
australian	30	83.73 ± 4.2	78.8 ± 8.2	84.18 ± 4.27	0	1
australian	50	83.71 ± 4.51	76.4 ± 10.51	84.64 ± 4.28	1	1
australian	100	83.72 ± 4.77	73.03 ± 13.17	83.79 ± 4.55	0	1
breast	1	95.75 ± 2.25	94.42 ± 2.87	95.46 ± 2.37	0	1
breast	10	95.48 ± 2.42	91.42 ± 3.71	94.89 ± 2.7	-1	1
breast	30	95.69 ± 2.54	89.12 ± 4.69	94.98 ± 2.42	-1	1
breast	50	95.83 ± 2.26	87.83 ± 5.84	95.14 ± 2.74	-1	1
breast	100	95.59 ± 2.44	83.2 ± 9.28	95.28 ± 2.44	0	1
german	1	68.14 ± 4.04	67.74 ± 4.38	67.72 ± 4.29	0	0
german	10	68.46 ± 4.29	66.72 ± 4.54	68.72 ± 4.5	0	1
german	30	68.77 ± 4.12	62.96 ± 6.14	69.18 ± 4.42	0	1
german	50	68.33 ± 4.45	62.81 ± 6.5	69.3 ± 4.85	1	1
german	100	67.98 ± 3.68	63.89 ± 6.59	68.13 ± 4.4	0	1
heart	1	77.67 ± 7.23	74.7 ± 8.22	77.0 ± 7.52	0	1
heart	10	75.78 ± 7.6	70.37 ± 9.18	75.81 ± 7.8	0	1
heart	30	76.59 ± 7.77	68.85 ± 11.21	75.78 ± 7.78	0	1
heart	50	77.74 ± 7.56	55.55 ± 0.0	76.52 ± 8.59	0	1
heart	100	76.81 ± 8.22	55.55 ± 0.0	76.89 ± 8.05	0	1
iris	1	94.07 ± 6.69	92.33 ± 7.05	93.53 ± 6.66	-1	1
iris	10	94.93 ± 6.08	84.07 ± 12.35	94.13 ± 6.3	-1	1
iris	30	94.93 ± 5.45	38.2 ± 12.81	94.53 ± 5.31	0	1
iris	50	94.8 ± 5.57	31.33 ± 7.96	94.07 ± 5.52	-1	1
iris	100	94.87 ± 5.51	32.33 ± 5.71	93.73 ± 5.98	-1	1
kr-vs-kp	1	99.48 ± 0.35	98.84 ± 0.65	99.41 ± 0.41	-1	1
kr-vs-kp	10	99.36 ± 0.47	95.31 ± 1.85	99.4 ± 0.42	0	1

Continua en la siguiente página

Cuadro 8.2 – continuación de la página previa

BD	#AgTrab	Certeza Cen	Certeza Base	Certeza Colab	ColvsCen	ColvsBas
kr-vs-kp	30	99.44 ± 0.42	92.02 ± 2.65	99.39 ± 0.44	0	1
kr-vs-kp	50	99.36 ± 0.47	89.11 ± 5.68	99.42 ± 0.43	0	1
kr-vs-kp	100	99.42 ± 0.4	78.7 ± 9.94	99.45 ± 0.38	0	1
letter	1	87.96 ± 0.82	83.66 ± 0.9	88.27 ± 0.73	1	1
letter	10	87.94 ± 0.71	69.24 ± 1.35	88.18 ± 0.77	1	1
letter	30	88.03 ± 0.74	57.74 ± 2.01	88.23 ± 0.87	1	1
letter	50	88.07 ± 0.71	51.12 ± 2.22	88.13 ± 0.69	0	1
letter	100	88.04 ± 0.79	39.83 ± 3.19	88.31 ± 0.69	1	1
poker	1	99.79 ± 0.02	99.69 ± 0.02	99.74 ± 0.05	-1	1
poker	10	99.80 ± .01	98.96 ± 0.12	99.56 ± 0.04	-1	1
poker	30	99.79 ± .02	95.80 ± 0.41	99.57 ± 0.03	-1	1
poker	50	99.78 ± .01	92.53 ± 1.07	99.33 ± 0.24	-1	1
poker	100	99.78 ± .01	87.36 ± 1.52	99.43 ± 0.32	-1	1
waveform	1	75.34 ± 1.98	74.66 ± 2.13	74.9 ± 2.03	0	0
waveform	10	75.27 ± 1.83	71.33 ± 2.53	74.74 ± 1.74	-1	1
waveform	30	75.21 ± 2.06	67.83 ± 3.01	75.2 ± 1.81	0	1
waveform	50	75.3 ± 1.96	64.75 ± 4.0	74.69 ± 1.81	-1	1
waveform	100	75.15 ± 1.99	62.71 ± 4.57	75.38 ± 2.02	0	1

Cuadro 8.3.: Comparación de tiempos (en milisegundos) sin utilizar pruning

BD	#AgTrab	Media tiempo Cen	Media tiempo Colab
adult	1	1304.22	5764.04
adult	10	1280.06	13267.82
adult	30	1325.15	14723.33
adult	50	1325.91	12868.01
adult	100	1349.8	10146.03
australian	1	13.23	26.16
australian	10	9.18	120.48
australian	30	4.13	282.73
australian	50	4.03	360.02
australian	100	4.0	477.86
breast	1	1.91	25.79
breast	10	2.23	93.55
breast	30	2.36	174.57
breast	50	2.47	223.31
breast	100	3.5	371.8
german	1	4.42	48.31
german	10	11.62	258.37

Continua en la siguiente página

Cuadro 8.3 – continuación de la página previa

BD	#AgTrab	Media tiempo Cen	Media tiempo Colab
german	30	6.19	378.88
german	50	11.74	402.08
german	100	5.56	470.1
heart	1	2.01	24.32
heart	10	2.57	118.91
heart	30	2.46	284.45
heart	50	2.4	228.51
heart	100	2.3	413.8
iris	1	6.89	10.17
iris	10	6.79	35.95
iris	30	1.9	73.4
iris	50	2.1	73.28
iris	100	2.0	153.86
kr-vs-kp	1	11.01	62.36
kr-vs-kp	10	13.59	155.91
kr-vs-kp	30	18.19	262.52
kr-vs-kp	50	13.38	268.16
kr-vs-kp	100	18.33	421.08
letter	1	697.29	5153.8
letter	10	695.38	15762.75
letter	30	706.0	16824.8
letter	50	736.35	13843.37
letter	100	742.83	11137.66
poker	1	105866.1	176452.4
poker	10	102200.8	92.416.5
poker	30	97233.3	55915
poker	50	93845.1	45181
poker	100	96021.6	75724.9
waveform	1	359.02	2621.91
waveform	10	362.67	8881.6
waveform	30	371.82	9327.22
waveform	50	377.57	6435.08
waveform	100	391.67	7056.34

Cuadro 8.4.: Comparación de instancias de entrenamiento utilizando pruning

BD	#AgTrab	#Ej BD	#Ej Entr Cen	#Ej Entr Base	#Ej Entr Colab
adult	1	48842	43957	21978	27468.85 ± 107.53
adult	10	48842	43957	3996	16121.1 ± 147.73
Continua en la siguiente página					

Cuadro 8.4 – continuación de la página previa

BD	#AgTrab	#Ej BD	#Ej Entr Cen	#Ej Entr Base	#Ej Entr Colab
adult	30	48842	43957	1417	15162.07 ± 142.62
adult	50	48842	43957	861	15403.52 ± 163.4
adult	100	48842	43957	435	16063.0 ± 221.61
australian	1	690	621	310	383.58 ± 9.28
australian	10	690	621	56	226.86 ± 8.93
australian	30	690	621	20	213.35 ± 8.28
australian	50	690	621	12	210.71 ± 8.64
australian	100	690	621	6	209.04 ± 8.15
breast	1	683	614	307	341.71 ± 8.06
breast	10	683	614	55	144.12 ± 8.49
breast	30	683	614	19	118.73 ± 7.4
breast	50	683	614	12	115.16 ± 9.23
breast	100	683	614	6	111.89 ± 8.85
german	1	1000	900	450	698.2 ± 15.68
german	10	1000	900	81	613.64 ± 13.94
german	30	1000	900	29	614.03 ± 16.2
german	50	1000	900	17	618.74 ± 15.94
german	100	1000	900	8	629.59 ± 16.1
heart	1	270	243	121	182.5 ± 6.31
heart	10	270	243	22	145.79 ± 8.77
heart	30	270	243	7	143.24 ± 8.13
heart	50	270	243	4	140.66 ± 7.63
heart	100	270	243	2	140.99 ± 8.22
iris	1	150	135	67	72.87 ± 2.48
iris	10	150	135	12	29.18 ± 4.01
iris	30	150	135	4	26.83 ± 3.9
iris	50	150	135	2	25.98 ± 4.23
iris	100	150	135	1	25.84 ± 3.33
kr-vs-kp	1	3196	2876	1438	1459.93 ± 7.88
kr-vs-kp	10	3196	2876	261	345.35 ± 11.56
kr-vs-kp	30	3196	2876	92	205.78 ± 10.46
kr-vs-kp	50	3196	2876	56	181.04 ± 12.62
kr-vs-kp	100	3196	2876	28	160.91 ± 9.7
letter	1	20000	18000	9000	11803.68 ± 164.3
letter	10	20000	18000	1636	8349.14 ± 217.9
letter	30	20000	18000	580	8259.17 ± 238.86
letter	50	20000	18000	352	8389.38 ± 227.43
letter	100	20000	18000	178	8628.1 ± 284.24
poker	1	829201	746280	373140	374100.0 ± 14.24

Continua en la siguiente página

Cuadro 8.4 – continuación de la página previa

BD	#AgTrab	#Ej BD	#Ej Entr Cen	#Ej Entr Base	#Ej Entr Colab
poker	10	829201	746280	67843	71419.5 ± 150.61
poker	30	829201	746280	24073	38988.5 ± 1750.09
poker	50	829201	746280	14632	48773.5 ± 994.89
poker	100	829201	746280	7388	81041.5 ± 1141.97
waveform	1	5000	4500	2250	3836.38 ± 33.4
waveform	10	5000	4500	409	3534.6 ± 34.54
waveform	30	5000	4500	145	3523.18 ± 34.12
waveform	50	5000	4500	88	3543.13 ± 34.5
waveform	100	5000	4500	44	3561.68 ± 37.2

Cuadro 8.5.: Comparación de certeza de clasificación utilizando pruning

BD	#AgTrab	Certeza Cen	Certeza Base	Certeza Colab	ColvsCen	ColvsBas
adult	1	86.0 ± 0.44	85.78 ± 0.48	86.32 ± 0.45	1	1
adult	10	85.97 ± 0.44	84.75 ± 0.57	86.22 ± 0.56	1	1
adult	30	85.99 ± 0.43	83.84 ± 0.73	86.25 ± 0.57	1	1
adult	50	86.02 ± 0.44	83.54 ± 0.89	86.28 ± 0.51	1	1
adult	100	85.98 ± 0.43	82.2 ± 1.58	86.3 ± 0.52	1	1
australian	1	86.53 ± 3.74	85.32 ± 4.21	86.42 ± 4.0	0	1
australian	10	86.36 ± 3.64	83.0 ± 4.97	86.37 ± 3.88	0	1
australian	30	86.34 ± 3.78	77.76 ± 10.15	85.99 ± 3.9	0	1
australian	50	85.92 ± 4.09	74.2 ± 11.71	85.82 ± 3.94	0	1
australian	100	85.97 ± 3.85	70.56 ± 13.33	85.94 ± 3.86	0	1
breast	1	95.69 ± 2.04	94.67 ± 2.78	95.8 ± 2.0	0	1
breast	10	95.73 ± 2.65	91.04 ± 3.53	95.44 ± 2.43	0	1
breast	30	95.92 ± 2.23	88.02 ± 4.92	95.12 ± 2.57	-1	1
breast	50	95.7 ± 2.33	87.57 ± 5.99	95.11 ± 2.38	-1	1
breast	100	95.81 ± 2.38	80.73 ± 10.66	95.5 ± 2.49	0	1
german	1	72.05 ± 3.73	71.33 ± 4.05	71.82 ± 4.02	0	0
german	10	71.57 ± 3.74	68.38 ± 3.81	71.73 ± 3.78	0	1
german	30	71.83 ± 4.11	68.14 ± 3.89	71.18 ± 4.0	0	1
german	50	71.75 ± 4.0	66.56 ± 5.56	71.51 ± 3.96	0	1
german	100	72.5 ± 3.73	65.36 ± 7.94	71.79 ± 4.09	-1	1
heart	1	78.81 ± 7.97	75.26 ± 8.68	76.3 ± 8.9	-1	0
heart	10	78.07 ± 7.25	71.44 ± 8.61	77.3 ± 7.49	0	1
heart	30	77.0 ± 7.52	66.11 ± 11.12	77.11 ± 7.56	0	1
heart	50	77.3 ± 7.78	55.55 ± 0.0	77.33 ± 7.74	0	1
heart	100	77.63 ± 7.84	55.55 ± 0.0	77.52 ± 7.28	0	1
iris	1	94.8 ± 5.24	92.73 ± 6.84	93.87 ± 5.42	-1	1

Continua en la siguiente página

Cuadro 8.5 – continuación de la página previa

BD	#AgTrab	Certeza Cen	Certeza Base	Certeza Colab	ColvsCen	ColvsBas
iris	10	94.6 ± 5.42	83.4 ± 11.78	93.8 ± 5.63	-1	1
iris	30	94.47 ± 5.19	37.13 ± 11.34	94.07 ± 5.6	0	1
iris	50	94.67 ± 5.61	31.0 ± 8.55	94.13 ± 4.95	0	1
iris	100	94.8 ± 5.65	32.0 ± 6.56	93.87 ± 5.74	-1	1
kr-vs-kp	1	99.39 ± 0.47	98.94 ± 0.62	99.34 ± 0.44	0	1
kr-vs-kp	10	99.41 ± 0.46	95.12 ± 1.81	99.42 ± 0.43	0	1
kr-vs-kp	30	99.45 ± 0.4	92.68 ± 2.54	99.49 ± 0.36	0	1
kr-vs-kp	50	99.4 ± 0.51	90.27 ± 5.49	99.42 ± 0.46	0	1
kr-vs-kp	100	99.39 ± 0.43	78.25 ± 10.29	99.48 ± 0.4	1	1
letter	1	87.98 ± 0.76	83.74 ± 0.87	88.18 ± 0.74	1	1
letter	10	88.07 ± 0.7	69.28 ± 1.35	88.26 ± 0.7	1	1
letter	30	87.96 ± 0.8	57.86 ± 1.69	88.23 ± 0.84	1	1
letter	50	88.09 ± 0.73	51.26 ± 2.23	88.26 ± 0.8	1	1
letter	100	88.02 ± 0.67	40.35 ± 3.11	88.26 ± 0.76	1	1
poker	1	99.78 ± 0.01	99.76 ± 0.01	99.79 ± 0.01	0	0
poker	10	99.78 ± 0.01	99.06 ± 0.11	99.74 ± 0.01	-1	0
poker	30	99.79 ± 0.01	96.47 ± 0.25	99.76 ± 0.01	-1	0
poker	50	99.79 ± 0.01	92.22 ± 1.36	99.33 ± 0.02	-1	0
poker	100	99.79 ± 0.01	87.99 ± 0.4	98.99 ± 0.79	-1	1
waveform	1	75.35 ± 1.87	74.77 ± 2.03	75.24 ± 1.75	0	1
waveform	10	75.36 ± 1.99	70.89 ± 2.22	75.08 ± 1.88	0	1
waveform	30	75.35 ± 1.9	67.52 ± 3.03	74.69 ± 2.09	-1	1
waveform	50	75.05 ± 1.74	65.44 ± 3.26	74.85 ± 1.94	0	1
waveform	100	75.21 ± 1.99	62.76 ± 4.54	74.99 ± 2.04	0	1

Cuadro 8.6.: Comparación de tiempos (en milisegundos) utilizando pruning

BD	#AgTrab	Media tiempo Central	Media tiempo Colab
adult	1	1393.97	5913.78
adult	10	1419.8	14191.26
adult	30	1435.85	15167.84
adult	50	1441.34	12626.48
adult	100	1465.65	9720.67
australian	1	3.18	32.2
australian	10	4.55	179.61
australian	30	9.2	273.16
australian	50	4.46	393.27
australian	100	5.24	500.18
breast	1	1.79	24.55

Continua en la siguiente página

Cuadro 8.6 – continuación de la página previa

BD	#AgTrab	Media tiempo Central	Media tiempo Colab
breast	10	7.29	85.05
breast	30	2.62	174.58
breast	50	2.48	239.38
breast	100	2.46	360.71
german	1	10.14	68.16
german	10	7.7	264.52
german	30	6.7	385.76
german	50	6.73	402.97
german	100	6.89	546.88
heart	1	11.84	15.49
heart	10	7.64	110.06
heart	30	7.74	276.65
heart	50	2.74	306.71
heart	100	2.7	458.98
iris	1	6.86	9.96
iris	10	6.92	18.8
iris	30	2.03	63.28
iris	50	2.11	97.44
iris	100	2.16	132.09
kr-vs-kp	1	18.42	60.93
kr-vs-kp	10	16.08	162.35
kr-vs-kp	30	15.96	247.86
kr-vs-kp	50	15.76	317.27
kr-vs-kp	100	17.55	451.98
letter	1	795.76	5435.1
letter	10	816.49	17850.55
letter	30	813.13	18120.53
letter	50	826.68	14723.98
letter	100	848.36	11643.36
poker	1	143236.0	180256.0
poker	10	147610.0	120582.0
poker	30	145595.0	53229.0
poker	50	148476.0	54364.5
poker	100	147646.0	54837.0
waveform	1	372.84	3330.27
waveform	10	370.02	9056.13
waveform	30	377.05	9371.32
waveform	50	390.79	6669.84
waveform	100	399.83	6933.9

9. Discusión

Como los resultados muestran, la aproximación presentada reduce el número de instancias de entrenamiento utilizadas para inducir el modelo final. Esto puede verse en todos los casos, habiendo casos donde la reducción es muy significativa, como es el caso de poker donde se utilizó hasta un 5 % de los ejemplos totales. En el peor de los casos, base de datos german, se utiliza hasta un 65 % de los ejemplos. En la mayoría de los casos, menos instancias son requeridas entre mayor sea el número de agentes trabajadores (aunque esto puede variar dependiendo de los parámetros experimentales escogidos), hasta que se alcanza un número mínimo de instancias, donde no importa cuantos trabajadores haya, el número de instancias de entrenamiento se mantiene. Parece existir una correlación entre el porcentaje total de instancias de entrenamiento que una base de datos en particular requiere y su certeza de clasificación máxima. Entre menor es la certeza de clasificación, mayor es el porcentaje de instancias de entrenamiento requeridas y vise versa. Así por ejemplo, la base de datos german, cuyos resultados de certeza de clasificación son pobres en todos los casos, también requiere de un mayor porcentaje de instancias de entrenamiento, mientras que una base de datos como kr-vs-kp cuyos resultados de certeza de clasificación son muy buenos, requiere menos instancias de entrenamiento en proporción. Esta correlación puede darnos indicios sobre la complejidad de una base de datos en particular y puede llegar a ser útil en un futuro para el análisis de bases de datos. Las desviaciones estándar presentadas también indican que la estrategia de aprendizaje colaborativa propuesta, así como el proceso experimental, son estables, puesto que no existen variaciones muy grandes.

Con respecto a certeza de clasificación, la estrategia presentada mantiene una certeza similar a la estrategia centralizada. Incluso en los casos donde, de acuerdo a los resultados de prueba T pareada, la estrategia propuesto es derrotada por la estrategia centralizada, la diferencia no es muy grande. También hay que considerar, como ya se mencionó en la sección 7 que no se utilizó la versión corregida de la prueba T pareada de dos colas, por lo que ciertos resultados pueden no ser del todo precisos (como por ejemplo el resultado arrojado para iris con 10 agentes sin pruning, donde la desviación estándar sugiere que no hay diferencias significativas).

Se observa que los experimentos con pruning activo presentan en general mejores resultados de certeza de clasificación, tanto para la estrategia propuesta como para la centralizada. Sin embargo, los resultados de prueba T son muy similares entre los experimentos con pruning activo y pruning no activo, por lo que se concluye que el pruning no es un factor que afecte al esquema propuesto. En bases de datos grandes (poker por ejemplo) el tener pruning activo reduce la eficiencia del sistema, esto debido al costo

mismo del pruning.

Los resultados de tiempo pueden ser tomados como la parte más débil de este trabajo. Sin embargo, como se ha mencionado antes, el enfoque principal del presente no es por el momento eficiencia sino más bien número instancias de entrenamiento utilizadas para inducir y certeza de clasificación. Por razones experimentales, se simuló un ambiente distribuido, pero a fin de cuentas los experimentos fueron ejecutados en un ambiente centralizado. Esto significa que, conforme el número de agentes crece, la carga de trabajo en la computadora también lo hace, y la consecuencia directa de este incremento en la carga de trabajo es mayor tiempo de proceso. Este tiempo es consumido en parte para administrar y controlar los diferentes hilos que cada agente representa. El tiempo también se incrementa al incrementar los agentes debido a que más fases del proceso social son requeridas y en consecuencia, más inducciones (en realidad se puede ajustar el momento de la inducción) e intercambios de comunicación son requeridos. Muchos de los problemas mencionados anteriormente pueden mejorarse significativamente al cambiar el esquema centralizado actual por uno distribuido que podría ser paralelo. Como muestra de lo mucho que se puede mejorar, algunos resultados con 100 agentes superan en eficiencia a otros con 30 o 50 agentes (como es el caso de la base de datos adult), siendo que en principio no debería ser posible, esto gracias a que fueron ajustados con parámetros que favorecieron la eficiencia en ese caso en particular (ver sección 6.5 para mas detalles sobre dichos parámetros). En realidad, si interesa mucho la eficiencia, puede jugarse con distintos parámetros que inevitablemente reducen la certeza de clasificación, pero que incrementan dramáticamente la eficiencia, en los experimentos presentes, se ajustaron dichos parámetros de forma empírica para que se arrojaran los mejores resultados de certeza de clasificación.

El esquema presentado tiene su claro costo, dada su naturaleza incremental. Este coste es más obvio cuando se compara con una inducción directa del modelo. Todo el proceso de búsqueda de ejemplos contradictorios y re-inducción de modelos es costoso. La mayoría de resultados aparentemente sugieren que es mejor utilizar la estrategia centralizada directamente ya que a pesar de que requiere de más ejemplos de entrenamiento es significativamente más rápida. Sin embargo, hay que considerar que la mayoría de bases de datos probadas son pequeñas, donde la inducción del modelo y la transmisión de datos no es costosa. En este tipo de escenarios, la estrategia centralizada puede ser utilizada directamente, pero en escenarios donde las bases de datos son verdaderamente grandes, hablando en términos de millones de ejemplos, y donde los datos se encuentran distribuidos geográficamente, la estrategia centralizada es inaplicable. El proceso de inducción y la transmisión de datos se vuelve muy costosa cuando el número de instancias es muy grande. El esquema propuesto reduce este problema, el número de ejemplos puede ser reducido significativamente, así que el costo en la transmisión de datos he inducción del modelo también se reduce. Así que, si por ejemplo, la propuesta presente realiza en total 100 inducciones, la cantidad de tiempo requerido para este número de inducciones puede ser menor al tiempo requerido para realizar una única inducción con

todos los datos. Una prueba de lo dicho anteriormente puede verse en los resultados para la base de datos poker. Dicha base de datos es grande (al rededor de 800 mil instancias), en una base de datos de este tamaño es posible ver cómo de hecho el esquema actual puede ser más eficiente que la estrategia centralizada y esto tomando en cuenta que el esquema propuesto puede ser mejorado bastante en el aspecto de eficiencia y que en realidad por la forma en que se realizaron los experimentos (en una sola computadora) el costo de transmisión de datos, que afecta mayormente a la estrategia centralizada, no está presente.

Con todo lo dicho, no es del todo claro si la aproximación general presentada puede contender actualmente contra estrategias más establecidas de minería de datos distribuida como es Meta-Aprendizaje, pero ciertamente los resultados obtenidos son prometedores y dan motivos suficientes para realizar una exploración y análisis más a fondo.

10. Conclusión y trabajo futuro

Como los resultados sugieren, la estrategia de aprendizaje colaborativo propuesta logra cumplir con la expectativa de reducir el número de ejemplos de entrenamiento necesarios para realizar una inducción mientras al mismo tiempo mantiene una certeza de clasificación similar a la de la estrategia centralizada. Esta estrategia es una forma prometedora para minería de datos distribuida y otros análisis de minería de datos.

Ya que fue posible corroborar la viabilidad, de este esta estrategia aplicada a minería de datos distribuida, ahora se está en posición de crear una solución más integral para minería de datos distribuida, una solución que utilice las ideas aquí presentadas y que al mismo tiempo realice los cambios necesarios para mejorar la eficiencia. Así, el trabajo futuro directo es crear y probar un modelo más eficiente, directamente un modelo que no sea secuencial como el aquí presentado.

Otras tareas que han quedado pendientes en el presente y que sería interesante revisar en un futuro son las siguientes:

- Llevar a cabo una análisis más profundo acera del fenómeno que ocurre en los resultados presentados, donde parece haber una correlación entre el porcentaje total de instancias de entrenamiento que una base de datos en particular requiere y su certeza de clasificación máxima.
- Realizar experimentos con bases de datos más grandes, del orden de los millones de instancias para verificar si las tendencias observadas en las bases de datos relativamente grandes probadas se en el presente se mantienen.
- Evaluar la especificidad de los resultados arrojados y no sólo la certeza de clasificación
- Evaluar la viabilidad del esquema propuesto como herramienta para el preprocesado de bases de datos
- Búsqueda de parámetros óptimos para cada base de datos.
- Evaluación e integración de diversos clasificadores además de J48 (C4.5)
- Realizar una extensión que permita tratar datos expresados en términos de lógica de primer orden

Bibliografía

Bache, K., y Lichman, M. (2013). *UCI machine learning repository.* Descargado de `http://archive.ics.uci.edu/ml`

Bernon, C., Cossentino, M., y Pavón, J. (2005). An overview of current trends in european aose research. *INFORMATICA-LJUBLJANA-*, *29*(4), 379.

Blockeel, H., y De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial intelligence*, *101*(1), 285–297.

Bordini, R. H., Braubach, L., Dastani, M., El FSeghrouchni, A., Gomez-Sanz, J. J., Leite, J., ... Ricci, A. (2006). A survey of programming languages and platforms for multi-agent systems. *INFORMATICA-LJUBLJANA-*, *30*(1), 33.

Bordini, R. H., Hübner, J. F., y Wooldridge, M. (2007). *Programming multi-agent systems in agentspeak using jason* (Vol. 8). Wiley-Interscience.

Bordinim, R. H., Dastani, M., Dix, J., y El Fallah Seghrouchni, A. (2009). Programming rational agents in GOAL. *Multi-agent programming: Languages, platforms and applications*, *2*, 3–37.

Bourgne, G., El Fallah Segrouchni, A., y Soldano, H. (2007). Smile: Sound multi-agent incremental learning. En *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems* (p. 38).

Bourgne, G., Fallah-Seghrouchni, A. E., y Soldano, H. (2009). Learning in a fixed or evolving network of agents. En *Proceedings of the 2009 ieee/wic/acm international joint conference on web intelligence and intelligent agent technology-volume 02* (pp. 549–556).

Bourgne, G., Maudet, N., y Pinson, S. (s.f.). Hypothesis refinement: Building hypotheses in an intelligent agent system.

Bourgne, G., Soldano, H., y Seghrouchni, A. E. F. (2010). Learning better together. En *Ecai* (Vol. 215, pp. 85–90).

Bratko, I. (2001). *Prolog: programming for artificial intelligence.* Addison-Wesley.

Bratman, M. E. (1987). Intention, plans, and practical reason.

Bratman, M. E., Israel, D. J., y Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational intelligence*, *4*(3), 349–355.

Clocksin, W. F., y Mellish, C. S. (2003). *Programming in prolog.* Springer Verlag.

Da Silva, J. C., Giannella, C., Bhargava, R., Kargupta, H., y Klusch, M. (2005). Distributed data mining and agents. *Engineering Applications of Artificial Intelligence*, *18*(7), 791–807.

Dastani, M. (2008). 2apl: a practical agent programming language. *Autonomous agents and multi-agent systems*, *16*(3), 214–248.

Dennett, D. C. (1989). *The intentional stance.* MIT press.

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J.-J., Sandhu, S., ... Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. *The American journal of cardiology*, *64*(5), 304–310.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., y Uthurusamy, R. (1996). Advances in knowledge discovery and data mining.

Fierens, D., Ramon, J., Blockeel, H., y Bruynooghe, M. (2005). A comparison of approaches for learning probability trees. En *Machine learning: Ecml 2005* (pp. 556–563). Springer.

Ghallab, M., Nau, D., y Traverso, P. (2004). *Automated planning: theory & practice*. Access Online via Elsevier.

Gorodetski, V., y Kotenko, I. (2002). The multi-agent systems for computer network security assurance: frameworks and case studies. En *Artificial intelligence systems, 2002.(icais 2002). 2002 ieee international conference on* (pp. 297–302).

Guerra-Hernández, A., El Fallah-Seghrouchni, A., y Soldano, H. (2005). Learning in bdi multi-agent systems. En *Computational logic in multi-agent systems* (pp. 218–233). Springer.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., y Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18.

Kersting, K. (2005). An inductive logic programming approach to statistical relational learning. En *Proceedings of the 2005 conference on an inductive logic programming approach to statistical relational learning* (pp. 1–228).

Kohavi, R., y cols. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. En *International joint conference on artificial intelligence* (Vol. 14, pp. 1137–1145).

Lehmann, E., y Deutsch, T. (1998). Compartmental models for glycaemic prediction and decision-support in clinical diabetes care: promise and reality. *Computer Methods and programs in Biomedicine*, *56*(2), 193–204.

Lloyd, J. W. (1984). *Foundations of logic programming* (Vol. 2). Springer-verlag Berlin.

Mitchell, T. M. (1997). Does machine learning really work? *AI magazine*, *18*(3), 11.

Neapolitan, R. E. (2004). *Learning bayesian networks*. Pearson Prentice Hall Upper Saddle River.

Okuyama, F. Y., Bordini, R. H., y da Rocha Costa, A. C. (2005). Elms: An environment description language for multi-agent simulation. En *Environments for multi-agent systems* (pp. 91–108). Springer.

Prodromidis, A., Chan, P., y Stolfo, S. (2000). Meta-learning in distributed data mining systems: Issues and approaches. *Advances in distributed and parallel knowledge discovery*, *3*.

Provost, F., y Domingos, P. (2003). Tree induction for probability-based ranking. *Machine Learning*, *52*(3), 199–215.

Quinlan, J. R. (1993). *C4. 5: programs for machine learning* (Vol. 1). Morgan kaufmann.

Rao, A. S. (1996). Agentspeak (l): Bdi agents speak out in a logical computable language.

En *Agents breaking away* (pp. 42–55). Springer.

Rao, A. S., y Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture.

Rao, A. S., Georgeff, M. P., y cols. (1995). Bdi agents: From theory to practice. En *Proceedings of the first international conference on multi-agent systems (icmas-95)* (pp. 312–319).

Rao, V. S. (2009). Multi agent-based distributed data mining: An overview. *International Journal of Reviews in Computing*, *3*, 83–92.

Ricci, A., Piunti, M., y Viroli, M. (2011). Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, *23*(2), 158–192.

Ricci, A., Viroli, M., y Omicini, A. (2006a). Construenda est cartago: Toward an infrastructure for artifacts in mas. *Cybernetics and systems*, *2*, 569–574.

Ricci, A., Viroli, M., y Omicini, A. (2006b). Programming mas with artifacts. En *Programming multi-agent systems* (pp. 206–221). Springer.

Russell, S. (2009). Artificial intelligence: A modern approach author: Stuart russell, peter norvig, publisher: Prentice hall pa.

Searle, J. R. (1962). Meaning and speech acts. *The Philosophical Review*, *71*(4), 423–432.

Shoham, Y. (1993). Agent-oriented programming. *Artificial intelligence*, *60*(1), 51–92.

Stolfo, S., Fan, W., Lee, W., Prodromidis, A., y Chan, P. (1997). Credit card fraud detection using meta-learning: Issues and initial results. En *Aaai-97 workshop on fraud detection and risk management.*

Stolfo, S., Prodromidis, A. L., Tselepis, S., Lee, W., Fan, D. W., y Chan, P. K. (1997). Jam: Java agents for meta-learning over distributed databases. En *Proceedings of the 3rd international conference on knowledge discovery and data mining* (pp. 74–81).

Stone, P. (2007). Intelligent autonomous robotics: A robot soccer case study. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *1*(1), 1–155.

Tate, A., Hendler, J., y Allen, J. (1994). *Readings in planning.* Morgan Kaufmann Publishers Inc.

Walter, F. E., Battiston, S., y Schweitzer, F. (2008). A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems*, *16*(1), 57–74.

Witten, I. H., y Frank, E. (2005). *Data mining: Practical machine learning tools and techniques.* Morgan Kaufmann.

Wooldridge, M., Jennings, N. R., y cols. (1995). Intelligent agents: Theory and practice. *Knowledge engineering review*, *10*(2), 115–152.

Zeng, L., Li, L., Duan, L., Lu, K., Shi, Z., Wang, M., ... Luo, P. (2012). Distributed data mining: a survey. *Information Technology and Management*, *13*(4), 403–409.