E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Generación de prototipos para clasificación en entornos Big Data

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Luis Iñiguez Jiménez

Directores: Mikel Galar Idoate

José Antonio Sanz Delgado

Pamplona, 30 de Junio de 2016

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

En primer a Mikel Galar y José Antonio Sanz, los directores de este proyecto, por todo el tiempo y dedicación que me han ofrecido a lo largo de este proyecto.

A los profesores de la universidad con los que tanto he aprendido por las buenas y a veces por las malas.

A mis compañeros de grado, con los que tanto he disfrutado y sufrido durante mi paso por la universidad y que nunca olvidaré.

A mis amigos y amigas que a pesar de la distancia y las circunstancias que casi siempre nos alejan siempre me han animado y apoyado cuando lo necesitaba.

A mi familia que siempre se ha interesado por mí y por cómo estaba y que siempre me han apoyado.

Por último y no menos importante a mi padre y a mi madre que siempre me han apoyado en todo lo que he hecho, que nunca han dudado de mí cuando muchos no creían que nunca podría llegar a donde he llegado y que siempre se han preocupado por mí y me han tenido siempre en sus pensamientos.

A todos vosotros.

Gracias.

RESUMEN

El objetivo del proyecto es utilizar las nuevas tecnologías del campo del BigData para crear un algoritmo que consiga reducir un dataset de clasificación compuesto por muchísimos ejemplos a unos pocos prototipos que los representen sin perder calidad para ayudar a los sistemas de clasificación a enfrentarse a estos datasets tan grandes.

Para lograrlo partiremos de un algoritmo base de generación de prototipos. Este algoritmo es CHI-PR, el cual se basa en el método de CHI para generar reglas difusas. Usando las reglas difusas agrega los ejemplos y genera prototipos. Con este algoritmo se realizaran varias iteraciones utilizando distintos modelos y configuraciones. Para probar la calidad del algoritmo se testeara utilizándolo sobre distintos datasets de clasificación.

Finalmente se realizaran una serie de comparativas para verificar que el algoritmo supera al algoritmo base y obtener una versión estable que pueda ser utilizada por la gran mayoría de datasets de clasificación.

Palabras clave:

- *Generación de prototipos*
- *BigData*
- *Clasificación*
- *Algoritmo de los k vecinos más cercanos*

Índice

Capítulo 1 INTRODUCCIÓN

En este proyecto se desarrolla un nuevo algoritmo de generación de prototipos centrado en el entorno de BigData. Para ello utilizaremos un algoritmo base y tecnologías especializadas en trabajar con grandes cantidades de datos. Este algoritmo está centrado a ayudar a los sistemas de clasificación en su trabajo.

Los sistemas de clasificación tienen la función de usar datos estructurados en forma de instancias o ejemplos con distintos tipos de atributos como unidad de información para aprender de ellos y ser capaces de saber clasificar correctamente futuras instancias o ejemplos. En la actualidad existen sistemas de clasificación muy buenos pero por lo general no están pensados en ser utilizados en entornos BigData.

BigData es el término usado para describir problemas que solo pueden ser resueltos en un tiempo razonable mediante una supercomputadora o un clúster de ordenadores.

En los últimos años han ido apareciendo tecnologías como la automatización de procesos, wereables, internet de las cosas, etc... Ante esta creciente cantidad de dispositivos conectados a redes que generan y procesan gran cantidad de información los problemas de computación ya no se pueden resolver solo con un ordenador. Ahora necesitan ser resueltos por ordenadores más potentes dándose así un incremento de la popularidad del BigData. Debido a esta creciente popularidad han aparecido distintas tecnologías para resolver los problemas BigData. Una de las tecnologías más utilizadas para BigData es Hadoop.

Apache Hadoop es un framework que soporta aplicaciones distribuidas bajo licencia libre. Permite trabajar con miles de nodos y peta bytes de datos. Está diseñado para ser usado tanto por un solo ordenador como por un clúster de miles de ordenadores ofreciendo cada uno su capacidad de cómputo y almacenamiento. Además ofrece resistencia ante caídas mediante detección y manejo de fallos en los ordenadores del clúster.

Hadoop incluye los siguientes módulos:

- **Hadoop Common**: Serie de servicios comunes que son usados por el resto de módulos.
- **HDFS**: Sistemas de ficheros distribuidos de hadoop, ofrece un manejo rápido de los datos.
- **YARN**: Framework encargado de organizar los recursos del clúster.
- **MapReduce**: Sistema diseñado para procesar en paralelo grandes cantidades de datos.

Hadoop ofrece muchos servicios y soluciones para resolver problemas en entorno BigData, por ello han aparecido múltiples tecnologías basadas en Hadoop que añaden nuevas características y funcionalidades. Una de las más famosas y que se ha usado en este proyecto es Spark.

Spark es un Framework diseñado para la computación en clústeres que provee de un interfaz de programación centrado en el uso de una estructura de datos llamada *Resilent Data Distributed* (RDD). Spark funciona por encima de hadoop, por tanto tiene todas las ventajas que ofrece hadoop en sistemas distribuidos.

Estas tecnologías mencionadas ayudan a los sistemas de clasificación a afrontar los problemas que da el trabajar en entornos BigData, sin embargo siguen siendo insuficientes. Por ello se utilizan algoritmos adicionales para ayudar a los sistemas de clasificación en su tarea. Este

proyecto se centra en crear uno de estos algoritmos para ayudar a los sistemas de clasificación. Los algoritmos que se han implementado entrarían dentro de la categoría de algoritmos de selección y agregación de prototipos.

Los algoritmos de selección y agregación de prototipos se encargan de analizar los datos, buscar las relaciones que hay entre ellos y agruparlos según características comunes para finalmente obtener un representante de estos al que se llama prototipo.

La selección de prototipos se basa en seleccionar las mejores instancias para eliminar el ruido e instancias que no son necesarias. La generación de prototipos se centra en crear nuevos prototipos que representen grupos de instancias para condensar la información.

Idealmente se pueden hacer algoritmos híbridos que combinen la selección de prototipos y la generación de prototipos para que el resultado sea de la mejor calidad posible.

Para realizar esta labor generalmente se utilizan distintas funciones de distancia que permiten saber cuan parecidos son los datos y funciones de medias para agrupar los datos más similares y así obtener los prototipos finales.

Para comprobar si se han generado buenos prototipos, una de las mejores maneras es usar el algoritmo kNN. Este algoritmo de clasificación se basa directamente en las instancias para clasificar siendo idóneo para ver como mejora tanto en precisión como en rapidez.

Los algoritmos que se han creado en este proyecto entrarían dentro de esta categoría. Sin embargo tienen la peculiaridad de que no utilizan funciones de distancia para agrupar instancias. Se ha utilizado el método de CHI.

El método de CHI es un sistema de clasificación basado en reglas difusas lo que lo convierte en un modelo interpretable. El método de CHI tiene dos principales componentes:

- **Base de conocimiento**: Está compuesta por una base de reglas y una base de datos donde las reglas y las funciones de pertenencia usadas para modelar las etiquetas lingüísticas son almacenadas.
- **Método de razonamiento difuso**: Es el mecanismo para clasificar ejemplos usando la información almacenada en la base de conocimiento.

En este proyecto usaremos el método de CHI para agrupar los ejemplos según la división del espacio de ejemplos que crea en base a las funciones de pertenencia que utiliza para modelar las etiquetas lingüísticas que componen los antecedentes de las reglas. La razón de utilizar el método de CHI es que este es mucho más rápido que utilizar comparaciones y distancias que obligan a hacer un excesivo número de cálculos. El método de CHI también ofrece la ventaja de que se adapta bien al paradigma MapReduce haciendo que aun sea más rápido si se usa en Hadoop.

Para realizar el proyecto se empleara el lenguaje de programación Scala. Scala es un lenguaje de programación orientado a objetos que también implementa el paradigma de programación funcional. Scala está basado en Java y por ello tiene todas las grandes ventajas de este. Corre sobre la JVM (Java Virtual Machine) y es compatible con aplicaciones Java. Como su propio nombre indica, la mejor característica de Scala es su escalabilidad. El lenguaje de programación Scala ofrece una sintaxis única que permite desarrollar aplicaciones rápidamente, dando facilidades en temas como la concurrencia, la programación distributiva y el reconocimiento de patrones que son de utilidad para la realización de este proyecto.

El objetivo del proyecto es utilizar las nuevas tecnologías referentes al ámbito del BigData para crear un algoritmo que consiga reducir la gran cantidad de ejemplos de un dataset a solo unos pocos prototipos sin perder la calidad para ayudar a los sistemas de clasificación a enfrentarse a este tipo de problemas. La realización del proyecto se ha dividido en diversas fases.

La primera fase del proyecto ha consistido en recopilar información sobre todo lo que se va a usar durante la realización del proyecto.

1. **Entorno BigData**: Este proyecto se ha realizado en un entorno BigData, por ello lo primero de todo ha sido aprender sobre él y sobre las herramientas que se necesitan. Las herramientas y tecnologías que se han estudiado han sido:
    - Hadoop y su uso en BigData.
    - El modo de funcionamiento de MapReduce.
    - Almacenamiento de grandes cantidades de datos mediante HDFS.
    - Spark, su manera de trabajar, de organizar los datos y el lenguaje en general.
2. **Programación en Scala**: Todo el proyecto ha sido desarrollado en Spark, Spark se puede implementar mediante diversos lenguajes: Python, Scala, Java y R. De entre todos los lenguajes se eligió Scala debido a su capacidad de permitir un desarrollo rápido y sencillo en poco tiempo, cualidad que comparte con Spark y que ha sido muy necesaria en este proyecto. Scala es un lenguaje que desconocía, por ello se ha realizado un estudio y aprendizaje del lenguaje. De Scala se ha puesto especial enfoque en sus estructuras de datos y sus distintos métodos de transformación y agregación de datos que se han usado continuamente debido al paradigma MapReduce que se emplea en BigData.
3. **Bases teóricas**: El proyecto se basa en utilizar el algoritmo CHI-PR. Aprender cómo se ha realizado y en que métodos se basa es fundamental para entender cómo funciona, implementarlo e intentar mejorarlo.

Tras la fase de preparación se llegó a la conclusión de que se podía desarrollar dos distintas implementaciones, una es un algoritmo de agregación al que se llamó CHI-PA y la otra es un algoritmo de refinamiento que se llamó CHI-PF. Todo el desarrollo y testeo inicial de los algoritmos se realizaron sobre una máquina local y no en un clúster de ordenadores. El desarrollo se realizó en diversos pasos.

1. **Entorno de desarrollo**: Lo primero que se necesito fue elegir un entorno de desarrollo para Scala y Spark. EL entorno de desarrollo elegido fue Intellij usando el sistema operativo Windows. Una vez instalado el entorno de desarrollo, tanto sistema operativo como IDE se configuraron para poder realizar aplicaciones Scala junto con los frameworks de Hadoop y Spark.

2. **Desarrollo de CHI-PR**: Antes de desarrollar los dos algoritmos que se plantearon, se implementó el algoritmo de CHI-PR. Debido a que ambos algoritmos utilizan CHI-PR la mejor opción fue implementar este algoritmo primero y comprobar su correcto funcionamiento para evitar futuros fallos.

3. **Desarrollo de CHI-PA**: El primer algoritmo que se desarrollo fue CHI-PA. De las dos versiones, CHI-PA es el que presenta una implementación más sencilla y que podía ser reutilizada luego por el otro algoritmo, CHI-PF. CHI-PA se basa en realizar sucesivas iteraciones de CHI-PR cambiando su configuración en cada iteración, por ello su implementación inicial resulto relativamente sencilla. Todo esto comprobando siempre su correcto funcionamiento.

4. **Desarrollo de CHI-PF**: Una vez desarrollado CHI-PA se desarrolló CHI-PF. Este algoritmo presenta una elaboración más costosa que CHI-PA debido a que este realiza procesos de refinamiento y filtrado de los ejemplos que llevaron en más tiempo de desarrollo.

5. **Mejora de las implementaciones**: Durante la implementación y testeo en un entorno local de ambos algoritmos se estudiaron varias opciones para mejorarlos. Por ello se realizó una etapa de mejora de ambos algoritmos antes de pasar a realizar pruebas en el clúster.

Una vez implementado los algoritmos se pasó de realizar pruebas en local a realizarlas en un entorno BigData, en un clúster. Para poder realizarlas hubo una fase de configuración del entorno de trabajo en el clúster y diversas pruebas y testeos de ejecución para comprobar el correcto funcionamiento de este. Una vez que se comprobó su correcto funcionamiento se pasó a testear los algoritmos. Debido a que los algoritmos están pensados a que funcionen de manera distribuida, sus ejecuciones cambiaban mucho de cuando se utilizaban sobre una sola maquina a cuando se utilizaron en el clúster que cuenta con varios nodos de ejecución. Por ello hubo una etapa extra de adaptación de los algoritmos al clúster.

Cuando ambos algoritmos se ejecutaron sin problemas en el clúster se pasó a una fase de mejora de ambas implementaciones. En esta fase se desarrollaron todas las ideas que surgieron durante la fase de desarrollo. Muchas de las ideas funcionaban bien cuando se probaban en un entorno local pero dejaban de funcionar a la hora de pasarlas al entorno BigData. Tras muchas pruebas se seleccionaron los resultados de las mejores implementaciones.

Capítulo 2 TÉCNICAS UTILIZADAS

En este capítulo veremos las distintas técnicas que se han usado en la realización de este trabajo. Primero se introducirá la clasificación dentro del campo de la minería de datos, luego varios conceptos del mundo del BigData, de ahí se pasarán a explicar distintas técnicas de clasificación y minería de datos que se han usado y finalmente se explicara CHI-PR, la técnica en la que se fundamenta el proyecto.

2.1 CLASIFICACIÓN

La clasificación es uno de los temas más estudiados dentro del campo de la minería de datos y el aprendizaje automático, la razón de esto es que hay una gran cantidad de problemas en diferentes áreas como seguridad, medicina o finanzas que necesitan clasificar muchos de los datos que manejan. El objetivo de los clasificadores es el de construir un modelo o clasificador a partir de un conjunto de ejemplos ya clasificados que permita clasificar nuevos ejemplos no vistos anteriormente en el futuro.

El problema al que nos enfrentamos en este proyecto es el de un problema de clasificación supervisada. Este tipo de problemas tienen dos fases principales:

1. El sistema de clasificación usará una serie de ejemplos llamados ejemplos de entrenamiento que ya estarán clasificados para aprender de ellos. Usando los datos de entrenamiento creará una serie de reglas o métodos de decisión para clasificar correctamente los ejemplos de entrenamiento.
2. Una vez creado el sistema de clasificación se pasara a clasificar ejemplos de test para ver cómo de bueno es el clasificador que hemos creado.

2.2 BIGDATA

Big data es un concepto que hace referencia al manejo de grandes cantidades de datos. Los problemas que podemos encontrar en entornos BigData son de lo más variopinto pero se pueden clasificar en unos pocos sectores que abarcan casi todos los problemas.

- **Problemas de captura y transformación**: Actualmente existe en el mundo un número cada vez más elevado de dispositivos conectados a la red. Todos ellos están continuamente mandando información, generando así una gran cantidad de datos de todos los ámbitos posibles. Ser capaz de capturar todos estos datos y de manejarlos en un tiempo asequible es uno de los problemas más típicos que se dan en el entorno BigData.
- **Problemas de almacenamiento**: En BigData no solo hay gran cantidad de datos sino que además pueden venir de diferentes sectores y en diferentes tipos de estructuras. Los datos pueden ser datos estructurados provenientes de bases de datos, datos semi-estrucutrados sacados de diferentes tipos de lenguajes de marcado o datos completamente desestructurados provenientes de cualquier programa o dispositivo. Como los datos pueden tener diferentes tipos de estructuras las bases de datos tradicionales se quedan obsoletas y hay que usar otros tipos de bases de datos que emplean diferentes técnicas.

    - **Bases de datos clave-valor**: Este tipo de almacenamiento se basa en crear mapas o diccionarios de datos mediante el uso de clave valores, un ejemplo de estas es Cassandra.
    - **Almacenamiento documental**: Este tipo de base de datos es muy similar a las de clave valor salvo por la diferencia de que estas tienen una estructura de datos específica para los valores siendo generalmente ficheros con formatos de lenguajes de marcado como XML, JSON u otros tipos. El más famoso de este tipo es MongoDB.
    - **Almacenamiento en grafo**: Es una forma diferente de relacionar grandes cantidades de datos pero usando la teoría de grafos como base para relacionar, las más conocidas son NEOJ4J y GraphDB.

- **Análisis de datos**: Analizar los datos es una tarea bastante común que se ve afectada por el hecho de tener que analizar grandes cantidades de datos. las tareas más comunes en este ámbito son:
    - **Clustering**: Técnicas que tienen la función de generar grupos de datos. Los grupos se generan de manera que los datos agrupados son muy parecidos entre sí pero muy diferentes de los datos de los otros grupos. Todo ello realizado de manera automática y sin supervisión alguna.
    - **Análisis de textos**: Extraer información que viene de fuentes que han sido creadas por personas como documentos, comentarios y correos electrónicos.
    - **Clasificación**: Técnicas que son usadas para realizar predicciones de a qué clase pertenecen diversos conjuntos de datos. Tratan de construir patrones mediante modelos y reglas.

Como se puede ver el BigData abarca muchas temáticas pero este proyecto se centra en el área de minería de datos usando las distintas tecnologías que nos ofrece Apache Hadoop.

2.2.1 Hadoop

Hadoop es un framework de código libre que se centra en la programación distribuida que sirve tanto para crear programas ejecutables en un solo ordenador como para correrlos en miles de computadoras de manera distribuida haciendo posible manejar grandes cantidades de datos y potencia de computo. Entre las ventajas que ofrece Hadoop están:

- **Accesibilidad**: Hadoop puede ser ejecutado tanto en una sola máquina, un clúster de ordenadores locales o en servidores remotos alojados en la nube dando así una flexibilidad que es difícil cuando se trata de ejecutar aplicaciones distribuidas.
- **Robustez**: Como Hadoop puede ser ejecutado en simples maquinas como ordenadores personales está diseñado para estar preparado para manejar todos los fallos que una computadora pueda dar.
- **Escalabilidad**: Al estar preparado para trabajar con miles de ordenadores en un clúster, añadir potencia de cálculo a aplicaciones exigentes es relativamente sencillo. Hadoop es capaz de añadir nodos de ejecución a esa aplicación en concreto si esta necesitase potencia de cómputo extra.
- **Simpleza**: Hadoop está diseñado para que se puedan escribir aplicaciones eficientes que se ejecuten en paralelo rápidamente cuando por lo general son bastante difíciles de hacer.

Como Hadoop es un framework especializado en procesar grandes cantidades de datos tiene ciertas características a la hora de trabajar que hay que tener muy en cuenta si se quiere realizar aplicaciones eficientes, de hecho, el proyecto que he realizado procura siempre utilizar estas características en su favor para ganar en eficiencia y en tiempo de ejecución.

- **Escalabilidad horizontal:** Una opción para obtener más potencia de cómputo es la de obtener grandes supercomputadores que puedan abordar todo tipo de problemas. Este tipo de arquitectura aunque es muy potente, es también muy costosa además de ser muy vulnerable frente a fallos de hardware. El tipo de arquitectura que utiliza hadoop es justo la contraria, en vez de utilizar supercomputadoras usa clústeres con miles de ordenadores con la que aporta flexibilidad a la hora de asignar recursos a diferentes aplicaciones y facilidad a la hora de aumentar la potencia de cálculo. Por tanto cuando usemos Hadoop hay que tener en cuenta que contaremos con muchos nodos repartidos entre varias máquinas, lo cual afecta en la manera en la que los datos se distribuyen por el clúster. Este proyecto ha sido ejecutado en un clúster de ordenadores, por tanto, obtenemos la ventaja de la ejecución distribuida que ofrece Hadoop.
- **Parejas clave-valor**: Hadoop trabaja con datos en estructuras de parejas clave-valor, de esta manera puede usar con mayor habilidad datos que vienen en formatos semi-estructurados o directamente sin estructurar. Esta manera de trabajar es beneficiosa para enfrentarse al problema recurrente del BigData de tener que tratar con datos semi-estructurados o sin estructurar. Esta filosofía de trabajar con los datos afecta a este proyecto de manera doble, por una parte utilizamos datos que vienen en un formato semi-estructurado, más concretamente en formato CSV (*Comma Separated Value*). Por otra parte utilizamos el método de CHI que tiene la característica de agrupar valores en parejas antecedente-ejemplos, método que abordaremos con más detenimiento más adelante.
- **Programación funcional y *MapReduce***: Debido a que los datos no están completamente estructurados las técnicas tradicionales para trabajar con datos no son completamente efectivas. Por ello se utiliza un paradigma de trata de datos llamado MapReduce que es pilar básico de Hadoop, debido a su importancia se explicará más adelante.
- **Datos de solo lectura**: Hadoop no es muy bueno a la hora de escribir muchos datos a la vez ya que se especializa más en trabajar con ellos, debido a esto es mejor trabajar con datos de solo lectura, analizarlos completamente y una vez terminado escribirlos. Esta manera de trabajar se complementa muy bien con el paradigma MapReduce.

2.2.1.1 HDFS

Hadoop Distributed File System es un sistema de ficheros distribuido diseñado para el procesado de ficheros de gran tamaño utilizando el framework MapReduce. Siendo un sistema distribuido no almacena todo en una sola máquina, lo almacena en varias, pudiendo de esta manera manejar ficheros del tamaño de 100TB. Además realiza varias copias en diferentes nodos para evitar problemas si un nodo se queda inoperativo. HDFS tiene las siguientes características clave:

- **Resistente a fallos**: HDFS tiene asumido que los fallos de hardware pasan, por ello distribuye los datos por todas las maquinas del clúster para que si alguna de ellas falla no afecte al resto del sistema además de tener un sistema de detección y recuperación de fallos.

- **Acceso a datos en flujo**: HDFS no está pensado para que sea usado directamente por personas, todo lo contrario, está diseñado para que lo utilicen aplicaciones y procesos. Por ello se centra más en dar prioridad a la tasa de transferencia frente a la latencia en el acceso a los datos.
- **Escribe una vez, lee muchas**: Las aplicaciones que usan HDFS tienden a escribir en ficheros una sola vez y a leer mucho de estas lo cual simplifica mucho los problemas de coherencia en los datos y un rápido flujo de estos.
- **Mueve la computación**: Cuando los datos son tan grandes se llega al punto en el que se vuelve más eficiente mover los procesos de computación hacia donde están los datos que mover los datos hacia la maquina donde está el proceso. HDFS provee de interfaces para que las aplicaciones sean capaces de realizar estas acciones.

La arquitectura de HDFS es de maestro esclavo. El NameNode es el servidor que maneja todos los metadatos de los ficheros y el acceso de los clientes a los datos. Los DataNodes son los esclavos encargados de guardar todos los bloques de información que le llegan. Cuando un cliente quiere almacenar un fichero en el HDFS, este da la apariencia de que el fichero esta entero en el sistema, pero internamente, el NameNode lo parte en bloques, los divide, los replica por todo el clúster y guarda toda la meta información del fichero localmente.

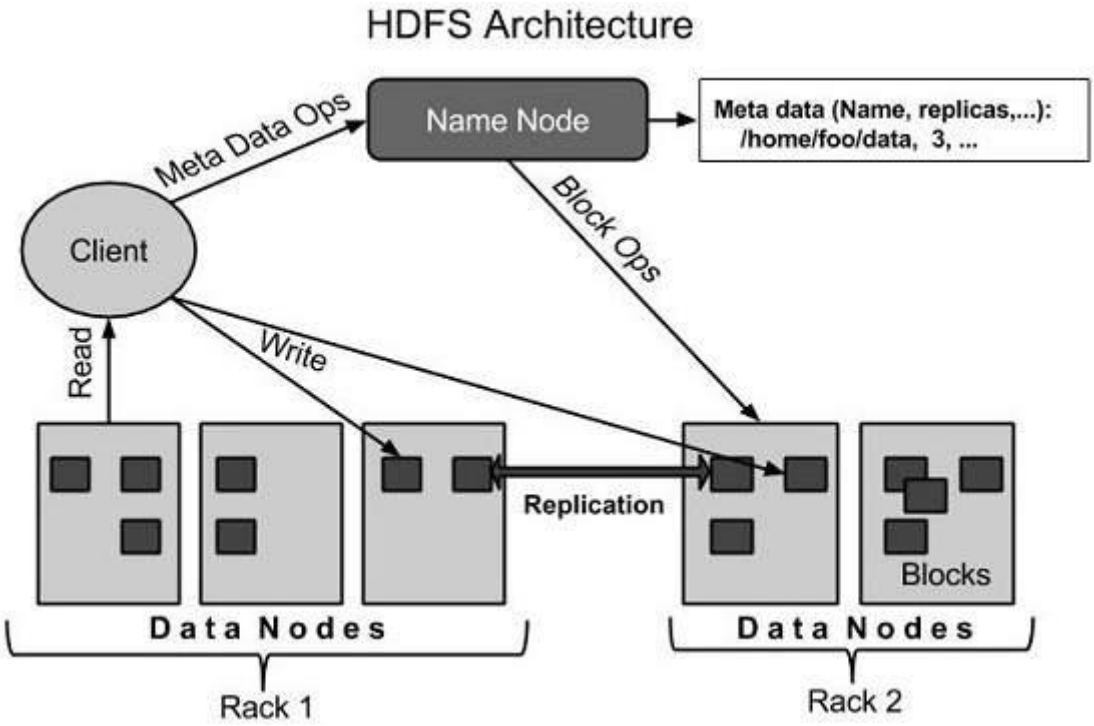

Figura 1. Arquitectura de Hadoop Distributed File System

2.2.2 Spark

Apache Spark es un framework de programación que trabaja por encima de hadoop ofreciendo varias ventajas. La principal ventaja de Spark y la razón por la que se usa en este proyecto es que es inmensamente más rápido que hadoop. Es rápido en ejecución y rápido en desarrollo. Además ofrece sencillez a la hora de realizar configuraciones.

Spark en si es un motor computacional. Es capaz de planificar, distribuir y monitorizar distintas aplicaciones que realizan distintas labores usando distintos trabajadores. De esta manera ofrece la posibilidad de crear aplicaciones complejas ayudando en el manejo y unificación de distintas tareas.

Spark viene integrado con múltiples componentes que todos juntos son los que permiten crear y organizar diferentes aplicaciones:

- **Spark core**: El núcleo de Spark contiene las funcionalidades básicas y contiene los componentes encargados de organizar tareas , controlar el uso de la memoria y el almacenamiento en disco además de tener la API que define los RDD, uno de los pilares de Spark del que se tratará más adelante
- **Clúster manager**: Al igual que Hadoop, Spark está pensado para que pueda ser usado desde un solo ordenador hasta un clúster con miles de ordenadores por eso puede ser ejecutado en una gran variedad de clústeres como YARN, MESOS o Cassandra.
- **Complementos**: Spark trae consigo distintos componentes dedicados a la trata de datos:
    - ***Spark SQL***: Paquete dedicado a trabajar con datos estructurados.
    - ***Spark streaming***: Paquete que permite el procesado de flujo de datos.
    - ***GraphX***: Librería específica para trabajar con grafos.
    - ***MLib***: Librería que contiene funcionalidades básicas pertenecientes al área del aprendizaje automático.

Una de las partes más importantes de Spark que se ha usado con frecuencia en el desarrollo de este proyecto es su estructura de datos RDD. Los RDD (*Resilent Data Distributed*) son la estructura básica de datos de Spark y casi todas las operaciones son de creación, transformación o llamadas de operaciones a RDD.

Los RDD son colecciones distribuidas de objetos inmutables divididos en diferentes particiones cada una de ellas pudiendo ser ejecutada en diferentes nodos, los objetos que puede contener pueden ser tanto de Python, Scala, Java o provenientes de clases definidas por el usuario.

Spark tiene la peculiaridad de que trabaja en modo lazy. Spark distingue entre dos tipos de operaciones, las transformaciones y las acciones. Cuando a Spark le llegue una instrucción de tipo transformación no la ejecutara inmediatamente, esperará. Cuando le llegue una operación de tipo acción entonces la ejecutará junto con todas las transformaciones anteriores que se ha guardado. Esta forma de trabajar permite a Spark planificar todas sus instrucciones de manera eficiente.

La unión del uso de los datos mediante la estructura RDD y la manera de organizar la ejecución de las instrucciones mediante su filosofía de trabajo lazy hacen a Spark inmensamente más rápido que hadoop.

2.2.3 MapReduce

MapReduce es un paradigma de programación utilizado para dar soporte a la computación paralela en el marco de Hadoop, el nombre viene del nombre de dos importantes funciones de la programación funcional, Map y Reduce.

Para usar un modelo en hadoop, este tiene que ser traducido al paradigma MapReduce. Por ello todas sus acciones han de tener una fase Map() y otra Reduce(). No todos los métodos

pueden adaptarse a la forma de trabajo MapReduce. Por ejemplo, los métodos iterativos no se adaptan bien a esta metodología, aunque Spark puede suplir esta carencia.

2.2.3.1 Map

El método Map se encarga de recoger los datos de entrada, dividirlos en bloques lógicos y dejarlos preferiblemente en el nodo donde estaban almacenados, de esta manera cada bloque puede ser procesado por una única unidad de proceso. Los datos de entrada son transformados en pares clave valor y son procesados por la función Map() definida por el usuario. Los encargados de ejecutar las operaciones de la fase Map son los mappers. El resultado de la operación son <k', v'> pares clave valor a los que se les denomina datos intermedios y son preparados para ser enviados a los reducers de la siguiente manera:

- Las salidas son ordenadas por claves y se crea por cada clave una lista de valores.
- Se selecciona un reducer por cada clave.
- Los datos intermedios son enviados a cada reducer.

2.2.3.2 Reduce

El reducer es responsable de agregar las salidas de los mappers y de reducir los datos según la función que se quiera aplicar. Cuando los reducers obtienen de los mappers todos los datos entonces es cuando aplica la función de reducción sobre la lista de valores asociada a una clave. De esta función salen todos los valores que se querían obtener de la operación MapReduce.

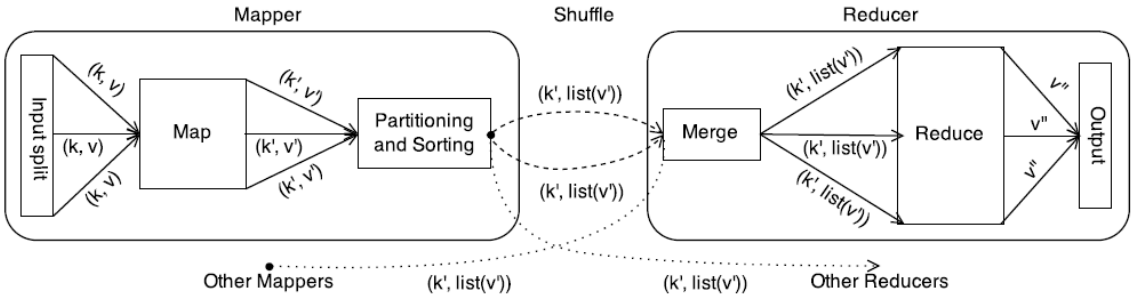

Figura 2. Representación del proceso MapReduce

2.3 KNN

Para saber cómo de buenos son los prototipos extraídos usaremos el algoritmo de clasificación kNN (*k Nearest Neighbours*). Este algoritmo clasifica cada instancia en base a sus vecinos. Dicho de otro modo, en base a las instancias que tenga más cerca y por tanto las más parecidas a esta. En el algoritmo kNN hay tres puntos fundamentales a decidir.

1. **El número de vecinos**: Afecta directamente a la cantidad de instancias que tenemos en cuenta, si son pocos podemos vernos afectados por instancias que son ruido, si son demasiados puede que clasifiquemos mal debido a que cojamos instancias de otras clases.
2. **El cálculo de la distancia**: La función de distancia es la que decide de entre todas las instancias cuales son las más cercanas. Las funciones de distancia más usadas son la distancia Euclídea y la distancia de Manhattan.

a. ***Distancia de Manhattan***: $d(x,y)\sum_{k=1}^{n}|x_k - y_k|$
b. ***Distancia Euclídea***: $d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$

3. **Método de voto**: una vez tenemos todos los ejemplos más cercanos hay que elegir de que clase es la instancia. Esta decisión se toma mediante distintos tipos de votaciones como votaciones simples o ponderadas.
    a. ***Votación simple***: Seleccionar los k ejemplos con la menor distancia al ejemplo que se quiere clasificar. El ejemplo a clasificar será de la clase que más se repite de entre los k ejemplos previamente elegidos.
    b. ***Votación ponderada***: La valía del voto viene en función de la distancia a la que se halla del ejemplo a clasificar.

$clase = \underset{v \in C}{\operatorname{argmax}} \sum_{y \in EK} \left(\frac{1}{d(e_y, e\prime)^2} * \left(v = clase(e_y)\right) \right)$ EK, conjunto de los k ejemplos más cercanos a e'.

Algoritmo 1. kNN con votación simple

Entrada: E, conjunto de datos de entrenamiento, x, ejemplo a clasificar, C, conjunto de clases posible del ejemplo y k, numero de vecinos.
Salida: $c \in C$ clase del ejemplo x

1. **Por cada** $e \in E$ **Hacer**
2. Calcular la distancia $d(e,x)$
3. **FIN**
4. Crear un subconjunto $N \in E$ con los k ejemplos de que más cercanos de x.
5. $clase_x = argmax_{c \in C} \sum_{n \in N} I(c = clase(n)$ donde $I(\cdot)$ es una función indicador que devuelve 1 si el argumento es verdadero o 0 en otro caso.

En este proyecto se ha usado para calcular la distancia entre dos ejemplos la distancia Euclídea y el método de voto la votación simple.

2.4 CHI

Los sistemas de clasificación basados en reglas difusas son unos de los métodos más comúnmente utilizados a la hora resolver problemas de clasificación. Estos algoritmos crean modelos interpretables por los humanos mediante el uso de reglas lingüísticas.

Los dos principales componentes de los sistemas de clasificación basados en reglas son:

- **Base de conocimiento**: Está compuesta por la base de reglas y la base de datos donde las reglas y las funciones de pertenencia usadas para modelar las etiquetas lingüísticas son almacenadas.
- **Método de razonamiento difuso**: Es el mecanismo para clasificar ejemplos usando la información almacenada en la base de conocimiento.

Para generar la base de conocimiento se usa un algoritmo de aprendizaje de reglas difusas en un dataset de entrenamiento compuesto por P ejemplos etiquetados $x_p = \left(x_{p1}, \dots, x_{pn}\right)$ con $p = \{1, \dots, P\}$ donde x_{pi} es el valor del i-ésimo atributo $(i = \{1, 2, \dots, n\})$ del p-ésimo

ejemplo de entrenamiento. Cada ejemplo pertenece a la clase $y_p \in C = \{C_1, C_2, \dots, C_m\}$ donde m es el número de clases del problema.

De entre todos los sistemas de clasificación basados en reglas difusas se eligió el algoritmo de Chi debido a que este se adapta bien al sistema de trabajo de *MapReduce*.

La estructura de reglas usada por este clasificador es la siguiente:

Regla R_j: Si x_1 es A_{j1} y… y x_n es A_{jn} entonces es de la clase C_j con RW_j

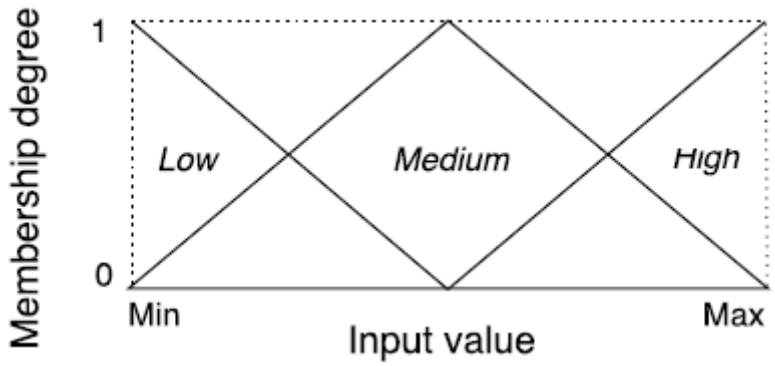

Figura 3. Ejemplo de variable difusa usando 3 etiquetas lingüísticas usando funciones de pertenencia triangulares.

Donde R_j es la etiqueta de la j-ésima regla, $x = \{x_1, \dots, x_n\}$ es el vector n-diemensional que representa al ejemplo, A_{ji}es la etiqueta lingüística modelada por una función de pertenencia triangular, C_j es la etiqueta de la clase y RW_j es el peso de la regla.

El factor de certeza penalizado es usado para calcular el peso de la regla:

$$RW_j = PCF = \frac{\sum_{x_p \in Clase\, C_j} \mu_{A_j}(x_p) - \sum_{x_p \notin Clase\, C_j} \mu_{A_j}(x_p)}{\sum_{p=1}^{P} \mu A_j(x_p)}$$

Donde $\mu A_j(x_p)$ es el grado de pertenencia del ejemplo x_p con el antecedente de la regla difusa R_j computada de la siguiente manera.

$$\mu_{A_j}(x_p) = \prod_{i=1}^{n} \mu_{A_{ji}}(x_{pi})$$

Siendo $\mu_{A_{ji}}(x_{pi})$ el grado de pertenencia del valor x_{pi} con el antecedente de la regla difusa R_j.

Para construir la base de reglas se sigue el siguiente algoritmo:

1. **Construcción de las etiquetas lingüísticas**: Los conjuntos difusos (etiquetas lingüísticas) son todas construidas con la misma forma triangular y distribuidas equitativamente en el rango de valores.
2. **Generación de reglas difusas por cada ejemplo**: Se genera una regla difusa por cada ejemplo x_p de la siguiente manera:
   a) Se computa el grado de pertenencia de cada variable de cada ejemplo x_p usando el correspondiente conjunto difuso.
   b) A cada variable se la asigna la etiqueta lingüística con mayor grado de pertenencia.
   c) El antecedente es determinado por la intersección de etiquetas lingüísticas y el consecuente es la etiqueta correspondiente a la clase del ejemplo.

d) El peso de la regla es computado usando la ecuación 2.

Con este algoritmo se puede dar el caso de que se generen reglas con mismo antecedente pero diferente consecuente, en estos casos se guardara únicamente la regla con mayor peso.

En el caso de que se den reglas con peso negativo estas serán eliminadas inmediatamente.

A la hora de clasificar un nuevo ejemplo usaremos el siguiente método:

1. **Grado de certeza**: Dado el ejemplo x_p calculamos su grado de pertenencia a cada regla de la base de reglas según la ecuación x.
2. **Grado de cobertura**: Se calcula por cada regla x_p de la siguiente manera $b_j(x_p) = \mu_{A_j}(x_p) \cdot RW_j$
3. **Clasificación**: Se predice la clase de la regla con el mayor grado de cobertura $Class = arg \max_{c=1,\dots,m} \left(\max_{R_j \in RB;\, C_j = c} b_j(x_p) \right)$

2.5 SELECCIÓN DE PROTOTIPOS

Los métodos de selección de prototipos son una parte de los métodos de selección de instancias cuyo principal objetivo es el de seleccionar al menor número posible de instancias que permitan a un algoritmo de clasificación aprender un modelo igual o más preciso que haciendo uso del dataset original. Minimizando el tamaño del dataset se consigue disminuir la complejidad y el coste computacional de los algoritmos de minería de datos.

De manera más formal, la selección de prototipos puede definirse de la siguiente: Sea $x_p$una instancia donde $x_p = (x_{p1}, x_{p2}, \dots, x_{pm}, x_{pc})$ con $x_p$perteneciente a la clase c dado por x_{pc} y un espacio m-dimensional donde x_{pi}es el i-ésimo valor de la p-ésima instancia. Entonces asumimos que hay un conjunto de entrenamiento TR con N instancias y un conjunto de test con T instancias. Finalmente se obtiene del algoritmo de selección de prototipos un subconjunto del conjunto de entrenamiento que será usado por el algoritmo de minería de datos. Para que la selección de prototipos sea efectiva, el algoritmo de minería de datos no tiene que perder precisión respecto al conjunto de entrenamiento original.

2.6 CHI-PR

Realizar tareas de minería de datos o de aprendizaje automático en entorno BigData es bastante costoso y dependiendo del tamaño del clúster que tengamos puedes ser hasta inviable, por ello las técnicas de pre procesado son importantes ya que limpian y simplifican los datos y hasta a veces ayudan a la precisión de los algoritmos. Generalmente los algoritmos de PR (*Prototype Reduction*) tienen un coste computacional $O(N^2)$ lo que en BigData los hace inviables. Sin embargo, CHI-PR, la técnica en la que se basa este proyecto, tiene un coste computacional de $O(N)$.

La idea principal detrás de CHI-PR es la de sacar ventaja de la buena compenetración del método de CHI con el paradigma MapReduce y la posibilidad de construir prototipos usando el sistema de reglas difusas.

2.6.1 Fase de mapeo: generación de antecedentes y agrupación de instancias

En la función Map() ejecutada por el mapper creamos una regla difusa para cada instancia. Cada instancia se mandará al combiner junto con su regla difusa. De la regla difusa cogeremos el antecedente, el cual se usara como clave del par <clave, valor> para agrupar las instancias.

Los antecedentes permiten dividir el espacio generado por los atributos en secciones, variando según el número de etiquetas lingüísticas usadas. Los antecedentes al estar formados por todos los atributos del problema, dividen la totalidad del espacio en múltiples celdas, cada celda tiene asignado un antecedente. De esta manera se consigue dividir el espacio y agrupar todas las instancias. Véase la figura 4.

El combiner recoge todos los datos, los ordena, agrupa por clave y las agrega. La manera de agregar las instancias agrupadas es sumándolas y contándolas en función de su clase. De esta manera tenemos por cada antecedente la suma de las instancias por clase junto con un contador para realizar posteriormente la media aritmética.

La estructura de datos final sería *<antecedents, (consequents, (numInstances, partialSum))>*, la clave seria el antecedente de la regla difusa y el valor contiene el consecuente de la regla y la suma y el conteo de todas las instancias que han generado esa regla.

Algoritmo 2. Función *map()* usada por el mapper en CHI-PR

Function *map* (key, value)
Input: <*key*, *value*> pair, representing the class and the values of the instance, respectively.
Output: <*key'*, *value'*> pair, where *key'* represents the antecedents of the generated rule and *value'* is a pair <*classIndex*, (1, *instanceValues*)>.
Begin
1: *ruleAnts* ← *generateRuleFromInstance* (*value*)
2: EMIT (*ruleAnts*, <*getClassIndex*(*key*), (1, *value*)>)
End

Algoritmo 3. Función *reduce()* usada por el combiner en CHI-PR

Function *reduce* (key, values)

Input: <*key'*, *values*> pair, where *key'* represents the antecedent part referring to the cell and *values* is a list containing <*classIndex*, (*numInstances*, *sumValues*)> pairs that store the count and the partial sum of those instances falling in the cell *key'* for each represented class.

Output: <*key"*, *value"*> pair, where *key"* is equal to *key'* and *value"* is a list of pairs in the same format as in *values*.

Begin

{Add the values of all the instances belonging to each class and count the number of instances of each class}

{*classSumValues* is a vector containing the sum of the values for each class. For nominal variables, it is replaced by a counter used to compute the mode in the Reducer.}

{*classNumInstances* is a vector containing the number of instances for each class }

```
1: while values.hasNext () do
2:     currentPair ← values.next ()
3:     currentClass ← currentPair.getClass ()
4:     currentValues ← currentPair.getValues ()
5:     for varIndex = 1 to NUM_VARIABLES do
6:         classSumValues[currentClass][varIndex] ← classSumValues[currentClass][varIndex] + currentValues[varIndex]
7:     end for
8:     classNumInstances[currentClass] ← classNumInstances[currentClass] + currentPair.getNumInstances ()
9: end while
```

{*listPairs* contains the count and the partial sum of all the instances belonging to each class.}

```
10: listPairs ← {}
11: for classIndex = 1 to NUM_CLASSES do
12:     if classNumInstances[classIndex] ≥ 1 then
13:         listPairs.add(<classIndex, (classNumInstances[classIndex], classSumValues[classIndex])>))
14:     end if
15: end for
```

{Send the list of pairs.}

```
16: EMIT (key', listPairs)
```

End

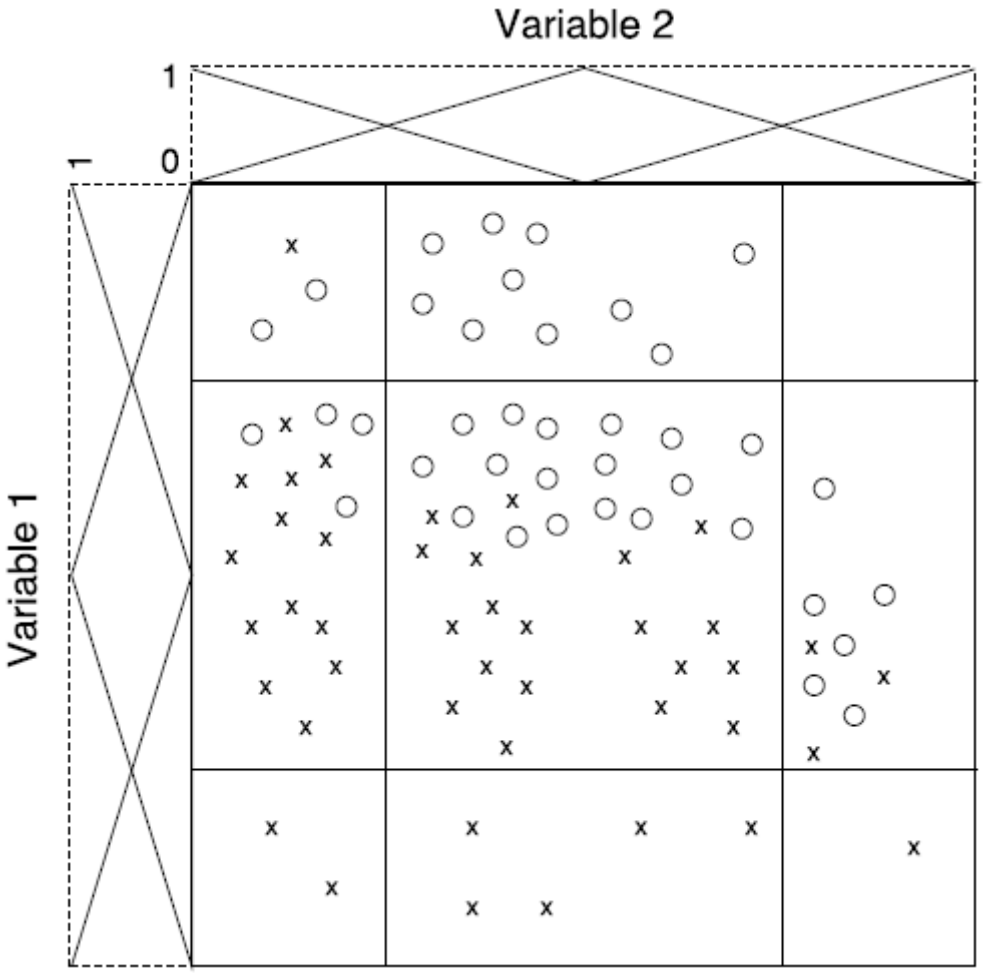

Figura 4. Particionado del espacio

En la figura 4 se ve de manera gráfica el resultado final de la fase de mapeo. En este ejemplo se tienen instancias compuestas por dos variables que pueden pertenecer a dos clases diferentes. Para generar las reglas difusas se han usado 3 etiquetas lingüísticas. De esta manera se divide el espacio en un total de 9 celdas. Las instancias caen en cada celda dependiendo del antecedente al que pertenecen y así poder ser agrupadas para formar uno o varios prototipos que representen a esa celda.

2.6.2 Fase de reducción: Obtención de prototipos

Una vez agrupadas, contadas y sumadas las instancias, ya se tiene toda la información necesaria para generar los prototipos, CHI-PR tiene dos versiones de generación de prototipos.

- **AM**: El reducer crea un prototipo de cada regla si tiene un mínimo número de instancias para evitar crear prototipos que generen ruido. El prototipo se genera usando la media aritmética de los valores de todos los ejemplos de cada clase. Figura 5.
- **SAM**: La generación de prototipos es idéntica que en el caso de AM. Sin embargo, por cada regla solo se saca el prototipo que ha sido generado con el mayor número de instancias. Esto significa que por cada regla solo se obtiene como máximo un prototipo. Figura 6.

Algoritmo 4. Función *reduce()* usada por el reducer en CHI_PR

Function *reduce* (key, values)

Input: <*key'*, *values*> pair, where *key'* represents the antecedent part referring to the cell and *values* is a list containing <*classIndex*, (*numInstances*, *sumValues*)> pairs that store the count and the partial sum of those instances falling in the cell *key'* for each represented class.

Output: <*key"*, *value"*> pair, where *key"* and *value"* are the values and the class of the prototype, respectively.

Begin

{Add the partial sum of all the instances belonging to each class and count the number of instances of each class. This step is the same as the one carried out in the Combiner.}

{*classSumValues* is a vector containing the sum of the values for each class. For nominal variables, it is replaced by a counter used to compute the mode.}

{*classNumInstances* is a vector containing the number of instances for each class }

1: **while** *values.hasNext* () **do**
2: *currentPair* ← *values.next* ()
3: *currentClass* ← *currentPair.getClass* ()
4: *currentValues* ← *currentPair.getValues* ()
5: **for** *varIndex* = 1 **to** *NUM_VARIABLES* **do**
6: *classSumValues*[*currentClass*][*varIndex*] ← *classSumValues*[*currentClass*][*varIndex*] + *currentValues*[*varIndex*]
7: **end for**
8: *classNumInstances*[*currentClass*] ← *classNumInstances*[*currentClass*] + *currentPair.getNumInstances* ()
9: **end while**

{Two variants of CHI-PR: CHI-PR$_{AM}$ and CHI-PR$_{SAM}$ }

10: **if** *CHI-PR*$_{AM}$ **then**
11: **for** *classIndex* = 1 **to** *NUM_CLASSES* **do**

{If there are enough instances belonging to this class, compute the arithmetic mean of those instances and create a new prototype. For nominal attributes, compute the mode instead of the arithmetic mean.}

12: **if** *classNumInstances*[*classIndex*] $\geq MinEx$ **then**
13: **for** *varIndex* = 1 **to** *NUM_VARIABLES* **do**
14: *classSumValues*[*classIndex*][*varIndex*] ← *classSumValues*[*classIndex*][*varIndex*] / *classNumInstances*[*classIndex*]
15: **end for**
16: EMIT (*classSumValues*[*classIndex*], *getClassLabel*(*classIndex*))
17: **end if**
18: **end for**
19: **else**

{Get the majority class}

20: *majClass* ← *getMajorityClass* (*classNumInstances*)

{If there are enough instances belonging to the majority class, compute the arithmetic mean of those instances and create a new prototype. For nominal attributes, compute the mode instead of the arithmetic mean.}

21: **if** *classNumInstances*[*majClass*] $\geq MinEx$ **then**
22: **for** *varIndex* = 1 **to** *NUM_VARIABLES* **do**
23: *classSumValues*[*majClass*][*varIndex*] ← *classSumValues*[*majClass*][*varIndex*] / *classNumInstances*[*majClass*]
24: **end for**
25: EMIT (*classSumValues*[*majClass*], *getClassLabel*(*majClass*))
26: **end if**
27: **end if**

End

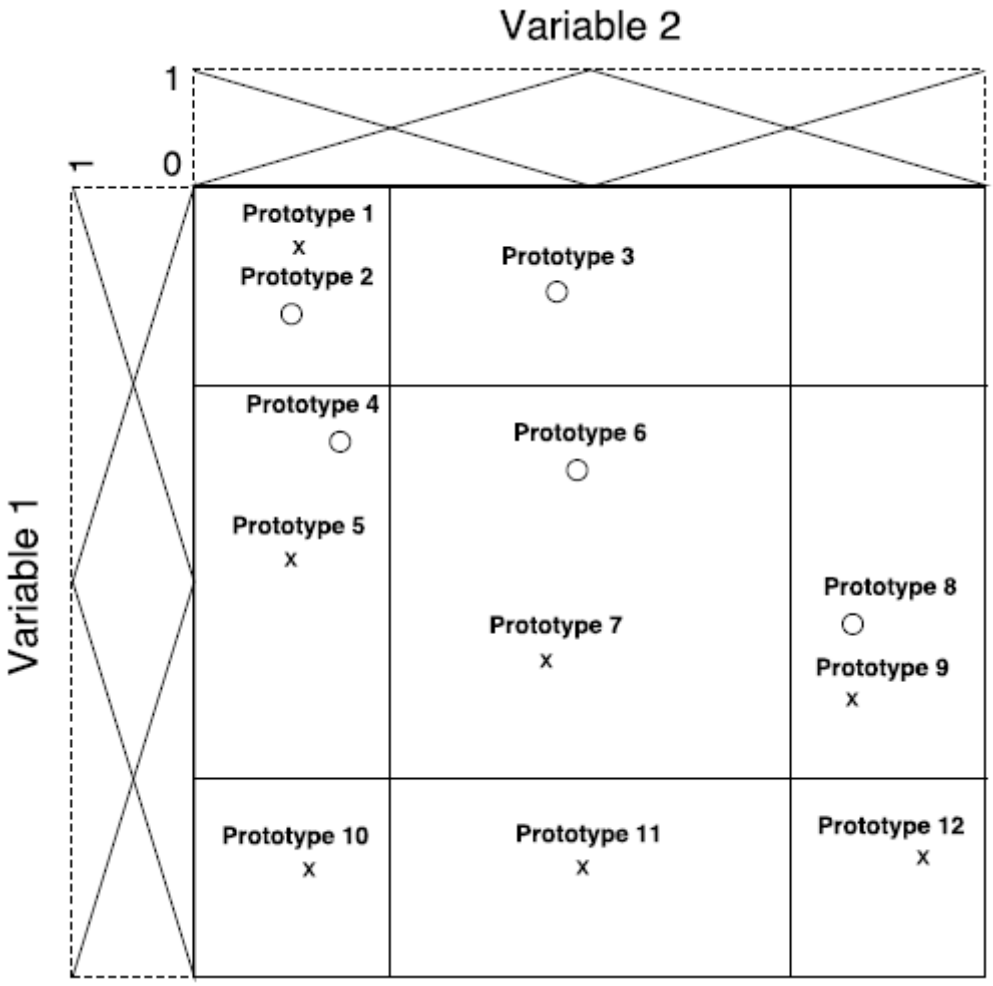

Figura 5. Generación de prototipos en CHI-PR (AM)

En la figura 5 se puede ver como se generan prototipos mediante el método AM. En cada celda se han generado uno o varios prototipos. Obviamente si no hay instancias en una celda no se puede generar prototipo.

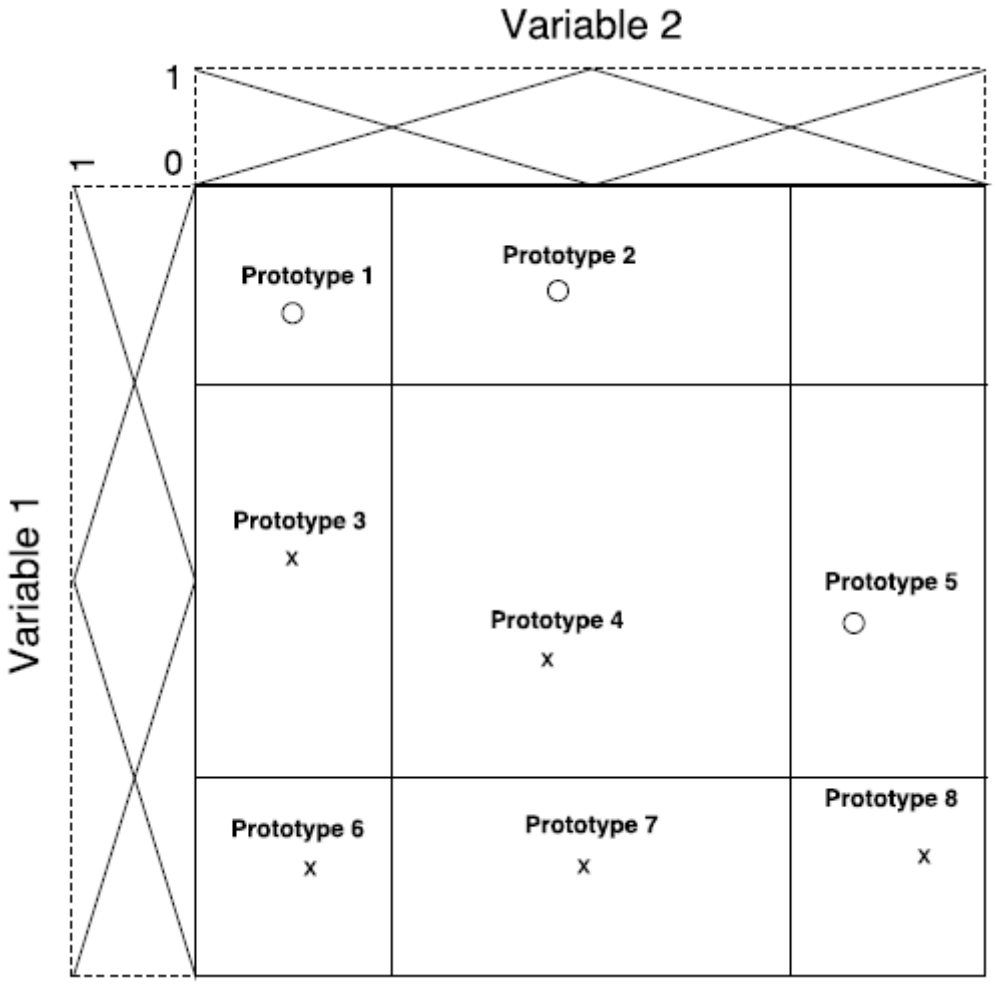

Figura 6. Generación de prototipos en CHI-PR (SAM)

En la figura 6 se han generado prototipos según el método SAM. Se puede apreciar comparándolo con la figura 4 y 5 como se ha generado como máximo solo un prototipo por cada celda. De poder salir más de un prototipo de una sola celda solo se genera el de la clase con mayor número de ejemplos.

Capítulo 3 MODELOS PLANTEADOS

En este capítulo se explicaran los distintos modelos que se han creado en este proyecto junto con los detalles de implementación, parámetros usados y la idea detrás de ellos.

La idea central de todas las versiones de los algoritmos es intentar mejorar la precisión de CHI-PR ejecutándolo varias veces usando distinto número de etiquetas lingüísticas. Usando distinto número de etiquetas lingüísticas se consigue que los antecedentes asociados a cada instancia cambien, variando la forma en la que se agrupan y generando distintos prototipos.

3.1 ALGORITMO DE REFINAMIENTO CHI-PF

Una de las claves de CHI-PR es la división del espacio del problema en celdas según los antecedentes. Cuantas más etiquetas lingüísticas se usan para generar antecedentes más celdas y de menor tamaño se crean. Antes de generar un prototipo a partir de las instancias de un antecedente se tiene que pasar una serie de condiciones para verificar si el prototipo que generan va a ser de calidad. Este algoritmo se centra en reutilizar las instancias que no han sido seleccionadas en la generación de prototipos para ser reutilizadas en iteraciones posteriores. En cada iteración se usa un número más alto etiquetas lingüísticas, dividiendo cada vez más el espacio y generando antecedentes más específicos para finalmente generar prototipos más refinados.

3.1.1 Generación de antecedentes y agrupación de instancias

Esta fase del algoritmo en su primera iteración es idéntica a la versión de CHI-PR, por cada instancia se genera una regla que corresponde a una mediante el uso de las etiquetas lingüísticas que mejor casan con el ejemplo. Finalmente se agrupan las instancias según el antecedente de la regla. De manera gráfica las instancias quedan contenidas en diferentes celdas.

3.1.2 Filtrado de instancias

En la fase de filtrado se decide que celdas son las que se van a seleccionar para generar prototipos y cuáles no. Las celdas se filtran en función de las instancias contenidas en ellas, para ello se utilizan combinaciones de varios métodos:

- **Mínimo número de ejemplos:** Al igual que en CHI-PR, en este algoritmo también filtramos según el número mínimo de ejemplos por antecedente. Lo que persigue este método de filtrado es que no se generen prototipos de antecedentes que solo tienen una o pocas instancias. Si se generasen prototipos de antecedentes con pocas instancias generaríamos demasiados prototipos que la mayoría serian ruido. Este método de filtrado es especialmente importante cuantas más etiquetas lingüísticas se han usado.
- **Media de instancias:** Se toma la media de instancias contenidas en las celdas. Las celdas que tengan contenidas un numero de instancias mayor que la media generaran antecedentes. Usando la media conseguimos separar las celdas que tienen mayor número de instancias de las que menos. Esta manera de filtrar funciona mejor cuando se usan pocas etiquetas lingüísticas para generar los antecedentes de las reglas y consecuentemente las celdas. Funciona mal cuando se han generado muchos antecedentes ya que el espacio se divide en muchísimas más celdas y la gran mayoría acaban con pocas instancias asociadas a ellas, en esta situación la media tiene a uno.

- **Porcentaje de instancias de una clase:** La idea de este método de filtrado es la de asegurarse de que el prototipo que se va a generar va a tener calidad. La manera de asegurarlo es calculando el porcentaje que hay de instancias de la clase dominante en una celda. Un porcentaje alto significa que casi todas las instancias de la celda son de la misma clase y si se genera un prototipo de estas no será ruido. Se han utilizado dos maneras.
    - **Porcentaje fijo:** Usando un porcentaje fijo podemos hacer que el proceso de filtrado sea todo lo restrictivo o permisivo que queramos, este porcentaje suele variar entre un 60% y un 90%. Tiene la desventaja de que tiende a funcionar peor cuando las instancias pueden ser de más de 2 clases dado a que el método se puede volver muy restrictivo.
    - **Porcentaje respecto de las clases:** El porcentaje a usar básico en este modo de filtrado es $1/num\ clases$. Este método es más flexible que el de porcentaje fijo ya que no tiene la problemática de los problemas multi-clase además de que puede ser configurado fácilmente para ser más restrictivo sumando pequeños porcentajes, generalmente con un rango $[10\%, 30\%]$.

Una vez que se han filtrado los antecedentes, los que han pasado el filtro pasan a la fase de generación de prototipos. Las instancias que han caído en las celdas que no han pasado el filtro se reservan para la nueva iteración que realizará el mismo proceso de generación de antecedentes y agrupación de instancias.

3.1.3 Generación de prototipos

Una vez filtrados los antecedentes se pasa a la fase de generar los prototipos. Al igual que en CHI-PR, los prototipos se pueden generar de dos maneras diferentes:

- **SAM:** Por cada antecedente contamos cuantas instancias hay de cada clase. Escogemos las instancias de la clase mayoritaria y realizamos la media aritmética. El resultado de la media es el nuevo prototipo. Con las instancias sobrantes no generamos prototipos ni tampoco se usaran para la siguiente iteración del algoritmo. La razón de que las instancias sobrantes no pasen a la siguiente fase es que son consideradas ruido. Si usásemos estas instancias en la siguiente iteración seguramente serian usadas para generar un nuevo prototipo. Este nuevo prototipo sería muy parecido al que acabamos de crear y perderíamos calidad.
- **AM:** Por cada antecedente comprobamos el número de instancias de cada clase, si hay un número mínimo generamos un prototipo de esa clase, pudiendo generar hasta un prototipo por clase, las instancias que no han sido usadas para generar prototipos no pasan a la siguiente iteración. Este método tiene sentido cuando los antecedentes han sido creados usando pocas etiquetas lingüísticas. Con pocas etiquetas lingüísticas las instancias se agrupan más y hay más posibilidades de que en un antecedente hayan varios prototipos claros.

Cuando se acaban de generar todos los prototipos se guardan. Todas las instancias de las celdas que han sido usadas para generar prototipos se eliminan.

3.1.4 Siguiente iteración

Las instancias a usar en esta iteración son las que guardamos en la fase de filtrado de la anterior iteración. De estas instancias solo necesitamos sus atributos y clase y volvemos a hacer todos los pasos descritos hasta ahora cambiando únicamente el número de etiquetas lingüísticas que se usaran para generar los nuevos antecedentes.

3.1.5 Salida del algoritmo

Una vez realizadas todas las iteraciones del algoritmo agregamos todos los prototipos generados en un único conjunto. El conjunto generado es el conjunto final de prototipos.

3.2 ALGORITMO DE AGREGACIÓN CHI-PA

La idea de este algoritmo se basa en agregar las salidas que dan múltiples ejecuciones de CHI-PR con diferentes configuraciones. Agregar salidas de CHI-PR hace que el número de prototipos final llegue a ser demasiado grande. Al igual que en la versión de refinamiento, se usaran diferentes filtros para reducir el número final de prototipos además de agregar una fase final de limpieza.

3.2.1 Generación de antecedente y agrupación de instancias

Esta fase no cambia nada respecto a otras versiones. Generamos por cada instancia un antecedente en función del número de etiquetas lingüísticas que usemos. Una vez creados los antecedentes agrupamos todas las instancias por antecedente o dicho de otro modo por celdas. Al igual que en el método anterior, las celdas pasan por una fase de filtrado antes de la de generación de prototipos.

3.2.2 Filtrado de instancias

Los métodos para filtrar las instancias son los mismos que en la versión de refinamiento:

- **Mínimo número de ejemplos:** Usado para evitar generar prototipos de antecedentes con un número insuficiente de instancias.
- **Media de instancias:** Usado para separar los antecedentes con pocas instancias de los que tienen muchas. Este método funciona mejor cuando se han usado pocas etiquetas lingüísticas para generar antecedentes.
- **Porcentaje de instancias de una clase:** Usadas para asegurarse de que el prototipo que se va a generar un prototipo de calidad.

En este método las instancias de las celdas que no han pasado el filtro en vez de reutilizarse para la siguiente iteración, directamente se desechan ya que en las próximas iteraciones se volverá a usar el conjunto de entrenamiento original.

3.2.3 Generación de prototipos

La generación de prototipos no varía de otras versiones. Una vez filtradas las instancias se pueden generar los prototipos de dos maneras diferentes.

- **SAM:** Por cada antecedente escogemos las instancias de la clase mayoritaria y realizamos con ellas la media aritmética. El resultado de la media es el nuevo prototipo.
- **AM:** Por cada antecedente agrupamos las instancias por clase. Generamos mediante la media aritmética un prototipo de cada clase si tienen un número mínimo de instancias asociadas. Los prototipos generados se guardan para ser agregados al final de las iteraciones.

3.2.4 Siguientes iteraciones

En la siguiente iteración aumentamos el número de etiquetas lingüísticas según la configuración que hayamos elegido. Los datos que se usaran para volver a generar prototipos será el conjunto de datos de entrenamiento original.

3.2.5 Fin del algoritmo

Al finalizar las iteraciones se agrupan todos los prototipos que se han generado en cada iteración creando el conjunto de prototipos total.

3.3 FASE DE LIMPIEZA

Debido a la alta cantidad de prototipos que se generan debido a que en ambos algoritmos se realizan varias iteraciones de CHI-PR, se hace necesario realizar una última fase de limpieza en ambas versiones.

La fase de limpieza es muy parecida en el modo de trabajar que los dos algoritmos planteados. Se harán $N - 1$ iteraciones donde N es el número de iteraciones que se han realizado para generar los prototipos. Un detalle a tener en cuanta en esta fase de limpieza es que el conjunto de número de etiquetas lingüísticas usadas en el método de generación de prototipos usado anteriormente se reutiliza. Por ejemplo, si se usaron $3, 5, y\ 7$ etiquetas lingüísticas en ese orden de iteraciones, para generar antecedentes y separar el espacio en celdas, en la fase de limpieza se utilizaran $5\ y\ 3$ etiquetas lingüísticas en ese orden.

3.3.1 Generación de antecedentes y agrupación de prototipos

Esta fase es igual que la de los métodos descritos. La idea es volver a dividir el espacio en celdas pero en vez de agrupar instancias se agrupan los prototipos generados. Una vez generados los antecedentes y dividido el espacio en celdas, agrupamos los prototipos. De esta manera se consigue separar los prototipos y crear conjuntos de prototipos que son muy parecidos entre ellos.

3.3.2 Limpieza de los prototipos por antecedente

Una vez agrupados los prototipos, cada antecedente tendrá asociados los prototipos que son más parecidos. De esta manera se pueden limpiar más fácilmente y comprobar cuáles son necesarios y cuáles no. En este proyecto se presentan dos formas de limpieza.

3.3.2.1 Limpieza mediante kNN

En esta fase se aplica la idea de kNN sobre los prototipos de cada antecedente por separado.

Una vez que se han agrupado los prototipos por celdas, realizamos operaciones individualmente por cada celda para limpiar los prototipos de ellas. Dada una celda escogemos un prototipo contenido en ella. Con el prototipo escogido calculamos su distancia con el resto de ellas. Una vez calculadas las distancias, escogemos los k prototipos con la distancia más corta al prototipo inicialmente escogido, dicho de otra manera, los k prototipos más parecidos. Si de esos k prototipos hay un número determinado que son de la misma clase que el prototipo escogido al principio, entonces eliminamos este del conjunto inicial de prototipos. Esta operación descrita se realiza con todos los prototipos de cada celda teniendo siempre en cuanta que si se elimina un prototipo durante el proceso, este no se utiliza para realizar las comparaciones con los otros prototipos.

Algoritmo 5. Limpieza mediante kNN.

Entrada: E, conjunto de prototipos, k, número de vecinos a comparar $umbral$, numero límite de prototipos parecidos.
Salida: conjunto E limpiado

1. $F = E$
2. **Por cada** $e \in E$ **Hacer**
    a. $F' = F - e$
    b. $D = \emptyset$
    c. **Por cada** $f \in F'$ **hacer**
        i. $D = D + distancia(e, f)$
    d. **Fin Por**
    e. $D' = \min(D, F', k)$
    f. **SI** $I(D', e, umbral)$ **entonces** $F = F - e$
3. **Fin Por**
4. $E = F$
5. **FIN**

$\min(D, F', k)$ Función que devuelve los k elementos de F' con las distancias asociadas D más bajas.
$I(D', e, umbral)$ Función booleana que devuelve verdadero si hay un número mayor o igual que $umbral$ de elementos de D' de la misma clase que e . Falso en caso contrario.

Tras la limpieza pasamos los prototipos que han quedado a la siguiente iteración.

3.3.2.2 Algoritmo de limpieza aleatorio

Este algoritmo sirve para hacer una limpieza breve en cada antecedente. Una vez agrupados los prototipos por celdas realizamos la misma operación por cada una de ellas. Escogemos k prototipos aleatoriamente. De los prototipos escogidos comparamos sus clases con el primero de ellos, si hay un número de ellos de la misma clase que el primero mayor que cierto umbral, eliminamos ese primer prototipo de la celda.

Algoritmo 6. Limpieza aleatoria.

Entrada: E, conjunto de prototipos asociados al mismo antecedente k, número de prototipos a comparar, $umbral$, numero límite de prototipos parecidos.
Salida: conjunto E limpiado

1. $F = escogerAleatorios(E, k)$
2. $e = primero(F)$
3. $contador = 0$
4. **Por cada** $f \in F$ **hacer**
    a. **SI** $e_{clase} = f_{clase}$ **entonces** contador = contador +1
5. **Fin Por**
6. **SI** $contador \geq umbral$ **entonces** eliminar el elemento e del conjunto E
7. **FIN**

La función $esocgerAleatorios(E, k)$ devuelve un subconjunto de E con k elementos aleatorios.

La función $primero(F)$ devuelve el primer elemento del conjunto F.

3.3.3 Siguientes iteraciones

En cada nueva iteración se utilizan los prototipos que han quedado tras la limpieza de la anterior iteración. Como se ha explicado anteriormente, hay que tener en cuenta el número de etiquetas lingüísticas que se van a usar para generar antecedentes y dividir el espacio en celdas en cada iteración y el número total de iteraciones que se van a realizar, $N - 1$.

3.3.4 Fin de la limpieza

Una vez que se han terminado todas las iteraciones, los prototipos que hayan quedado tras la fase de limpieza son los prototipos finales del algoritmo.

Capítulo 4 MARCO EXPERIMENTAL

En este capítulo se explicara los métodos y datasets que se han utilizado para evaluar el algoritmo.

4.1 MÉTODOS UTILIZADOS

Todos los métodos utilizados son variantes de los algoritmo propuestos que cambian según su configuración. Debido a que los mejores resultados de CHI-PR fueron con su configuración SAM con $mínimo\ de\ ejemplos\ =\ 3$, todos los experimentos han usado esta configuración.

En todas las variantes de los dos algoritmos presentados se han usado $3, 5\ y\ 7\ etiquetas\ linguisticas$ para generar antecedentes. Consecuentemente todas las configuraciones realizan 3 iteraciones. En las fases de limpieza se utilizan 2 iteraciones, $N-1$ iteraciones realizadas por el método de generación de prototipos, para limpiar los prototipos. Se utilizaran $5\ y\ 3\ etiquetas\ linguisticas$, en ese orden.

En las variantes en las que se utiliza el algoritmo de kNN como método de limpieza, la configuración usada ha sido: $k = 3, umbral = 3$.

Una vez generados los prototipos, estos se testean usándolos en un algoritmo kNN que emplea la siguiente configuración: $k = 1$, la distancia Euclídea y como método de voto la votación simple.

4.1.1 Algoritmo de agregación CHI-PA

Todas las configuraciones de los algoritmos de agregación utilizan siempre el filtrado de porcentaje respecto al número de clases. $porcentaje = {}^{1}/_{\#clases} + [0, 0.1, 0.2, 0.3]$.

- Algoritmo de agregación sin media y sin limpieza
- Algoritmo de agregación sin media y con limpieza kNN
- Algoritmo de agregación sin media y con limpieza aleatoria
- Algoritmo de agregación con media y sin limpieza
- Algoritmo de agregación con media y con limpieza kNN
- Algoritmo de agregación con media y con limpieza aleatoria

4.1.2 Algoritmo de filtrado CHI-PF

Todas las configuraciones de los algoritmo de filtrado utilizan siempre el filtrado de porcentaje fijo.$porcentaje = [0.8, 0.9]$. Este porcentaje decrece en un 0.1 en cada iteración que se realiza para que el algoritmo no se vuelva excesivamente restrictivo.

- Algoritmo de filtrado sin media y sin limpieza
- Algoritmo de filtrado sin media y con limpieza kNN
- Algoritmo de filtrado sin media y con limpieza aleatoria
- Algoritmo de filtrado con media y sin limpieza
- Algoritmo de filtrado con media y con limpieza kNN
- Algoritmo de filtrado con media y con limpieza aleatoria

4.2 DATASETS

Los datasets que se han utilizado para probar el algoritmo son los mismos que se han utilizado en CHI-PR. Utilizar los mismos datasets permitirá comparar el funcionamiento de ambos algoritmos.

Los experimentos con los datasets se han realizado usando la *validación cruzada en 5 subconjuntos.* Los datasets se parten en 5 particiones, cada partición tiene el 20% de las instancias. El conjunto de datos de entrenamiento se obtiene juntando 4 particiones. El conjunto de datos de test es el conjunto sobrante. Por combinatoria obtenemos finalmente 5 parejas de conjuntos $< entrenamiento, test >$.

Debido a que el tiempo para realizar este proyecto ha sido corto en comparación con la cantidad de tiempo necesario para realizar todos los experimentos, de las cinco particiones en las que se han partido los dataset, solo se ha usado una de estas para realizar los prototipos y por consiguiente todos los experimentos.

Dataset	#Ejemplos	Real	Entero	Nominal	Total
Covtype	581012	10	0	44	54
Poker	1025009	0	10	0	10
Skin	245057	0	3	0	3
Susy	5000000	18	0	0	18

4.3 MEDIDAS DE EVALUACIÓN

Para saber cómo de bueno es el algoritmo empleamos diferentes métodos de evaluación. La evaluación se empleara tanto en los modelos planteados como en CHI-PR.

4.3.1 Calidad de los prototipos

Lo primero y principal es conocer la calidad de los prototipos. Para saber la calidad empleamos el algoritmo kNN tanto en los algoritmos creados como en CHI-PR. La métrica utilizada para evaluar es el ratio de precisión.

$$Ratio\ de\ precision = \frac{Numero\ de\ ejemplos\ clasificados\ correctamente}{Numero\ total\ de\ ejemplos}$$

4.3.2 Tasa de reducción

Uno de los aspectos clave de la generación de prototipos es el número total de prototipos que se han creado. La manera más clásica de evaluar el número de prototipos generados es mediante el ratio de reducción.

$$Ratio\ de\ reduccion = \left(1 - \frac{Numero\ de\ prototipos\ generados}{Numero\ de\ instancias\ en\ el\ conjunto\ original}\right) \cdot 100$$

4.4 INFRAESTRUCTURA UTILIZADA

La infraestructura utilizada en este proyecto es la siguiente: Todos los algoritmos han sido ejecutados paralelamente por 7 nodos conectados entre sí en una LAN a una velocidad de 1Gb/s**.** Cuatro nodos tiene 2 procesadores Intel Xeon E5-2620 v3 con 2.4 GHz (3.2 GHz con Turbo Boost), cada uno con 12 núcleos virtuales (6 de ellos físicos). 2 nodos tienen 2

procesadores Intel Xeon E5-2620 v2 con 2.1GHz con el mismo número de núcleos que el anterior. El nodo restante es el maestro, el cual tiene un procesador Intel Xeon E5-2609 con 4 núcleos físicos a 2.4 GHz. El nodo maestro cuenta con 8GB de memoria RAM, los nodos maestros tienen 32GB de memoria cada uno.

Todos los nodos cuentan con discos duros que realizan operaciones de lectura/escritura a una velocidad de 128 MB/s.

Todo el clúster utiliza CentOS 6.5 y utiliza Apache Hadoop 2.6.0. Con una configuración que permite correr 42 contenedores YARN que pueden ser ejecutados como mappers, reducer o master.

Capítulo 5 ESTUDIO EXPERIMENTAL

El proyecto utiliza como base el algoritmo de CHI-PR para obtener uno nuevo que permita mejorar la precisión que dan los prototipos generados sin que pierda en exceso en el ratio de reducción. Por ello a la hora de contrastar resultados se utilizaran los resultados que ofrece CHI-PR como referencia. Para testear los prototipos se hará uso del algoritmo kNN con la misma configuración que utiliza CHI-PR para que las comparaciones sean más fiables.

En primer lugar se presentaran los datos respecto a la precisión obtenida comparándola tanto con los resultados de varias configuraciones de CHI-PR como con el resultado obtenido usando el algoritmo kNN usando todo el dataset.

En segundo lugar se compararan los ratios de reducción obtenidos con los del algoritmo CHI-PR usando distintas configuraciones.

Las versiones de CHI-PR con las que se van a comparar los resultados son las siguientes [1]

- **Mejor configuración**: Los resultados que da la mejor configuración de CHI-PR, la versión seria: $CHI - PR\ (SAM), 4\ etiquetas\ ling$üí$sticas,\ minExamples = 3.$
- **Mejores resultados**: los mejores resultados que ofrece CHI-PR por cada dataset independientemente de la configuración.
- **3, 5, 7 etiquetas**: Los resultados obtenidos usando $3, 5\ y\ 7\ etiquetas\ linguisticas$ y mínimo número de instancias para generar un prototipo en una celda = 3. Tomamos de referencia estas versiones ya que las mismas configuraciones han sido usadas tanto por CHI-PA y CHI-PF.

5.1 RESULTADOS DE CHI-PA EN PRECISIÓN

En este apartado se muestran los resultados en precisión que ha dado el algoritmo kNN usando los prototipos generados por CHI-PA en distintas configuraciones.

Los resultados están divididos en 3 tablas según método de limpieza siendo sin limpieza Tabla 1, con limpieza kNN tabla 2 y con limpieza aleatoria tabla 3.

Los resultados de cada configuración están ordenados por tablas distinguiéndose en cada una las versiones que han usado la media para filtrar las celdas de las que no. En primer lugar aparecen las distintas configuraciones de CHI-PA, luego diversas configuraciones de CHI-PR y finalmente los resultados que da el algoritmo kNN usando todo el dataset de entrenamiento original.

En cada columna se puede distinguir cual ha sido el mejor método, señalado en verde, y el peor señalado en rojo.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PA 0%	0.8741	0.8904	0.5355	0.5356	0.9630	0.9943	0.7320	0.7358
CHI-PA 10%	0.8743	0.8905	0.5355	0.5359	0.9583	0.9969	0.7344	0.7386
CHI-PA 20%	0.8741	0.8906	0.5358	0.5358	0.9583	0.9967	0.7348	0.7393
CHI-PA 30%	0.8736	0.8902	0.5366	0.5359	0.9583	0.9969	0.7303	0.7348
Mejor conf. CHI-PR	0.8087	0.8087	0.5249	0.5249	0.9893	0.9893	0.7215	0.7215
Mejores CHI-PR	0.8930	0.8930	0.5350	0.5350	0.9986	0.9986	0.7302	0.7302
CHI-PR 3 etiquetas	0.7749	0.7749	0.5334	0.5334	0.9753	0.9753	0.6877	0.6877
CHI-PR 5 etiquetas	0.8474	0.8474	0.5061	0.5061	0.9868	0.9868	0.7302	0.7302
CHI-PR 7 etiquetas	0.8832	0.8832	0.4879	0.4879	0.9943	0.9943	0.7281	0.7281
Full kNN	0.9428	0.9428	0.5087	0.5087	0.9996	0.9996	0.6934	0.6934

Tabla 1 Precisión en CHI-PA sin limpieza.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PA 0%	0.8532	0.8673	0.5357	0.5354	0.9650	0.9943	0.7102	0.7127
CHI-PA 10%	0.8531	0.8682	0.5355	0.5355	0.9530	0.9956	0.7135	0.7164
CHI-PA 20%	0.8525	0.8666	0.5356	0.5360	0.9555	0.9947	0.7145	0.7169
CHI-PA 30%	0.8524	0.8670	0.5360	0.5368	0.9555	0.9925	0.7096	0.7120
Mejor conf. CHI-PR	0.8087	0.8087	0.5249	0.5249	0.9893	0.9893	0.7215	0.7215
Mejores CHI-PR	0.8930	0.8930	0.5350	0.5350	0.9986	0.9986	0.7302	0.7302
CHI-PR 3 etiquetas	0.7749	0.7749	0.5334	0.5334	0.9753	0.9753	0.6877	0.6877
CHI-PR 5 etiquetas	0.8474	0.8474	0.5061	0.5061	0.9868	0.9868	0.7302	0.7302
CHI-PR 7 etiquetas	0.8832	0.8832	0.4879	0.4879	0.9943	0.9943	0.7281	0.7281
Full kNN	0.9428	0.9428	0.5087	0.5087	0.9996	0.9996	0.6934	0.6934

Tabla 2 Precisión en CHI-PA con limpieza kNN.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PA 0%	0.8732	0.8896	0.5358	0.5358	0.9629	0.9877	0.7312	0.7352
CHI-PA 10%	0.8727	0.8897	0.5355	0.5356	0.9571	0.9961	0.7339	0.7378
CHI-PA 20%	0.8732	0.8896	0.5356	0.5359	0.9558	0.9965	0.7346	0.7390
CHI-PA 30%	0.8725	0.8891	0.5361	0.5362	0.9548	0.9967	0.7298	0.7344
Mejor conf. CHI-PR	0.8087	0.8087	0.5249	0.5249	0.9893	0.9893	0.7215	0.7215
Mejores CHI-PR	0.8930	0.8930	0.5350	0.5350	0.9986	0.9986	0.7302	0.7302
CHI-PR 3 etiquetas	0.7749	0.7749	0.5334	0.5334	0.9753	0.9753	0.6877	0.6877
CHI-PR 5 etiquetas	0.8474	0.8474	0.5061	0.5061	0.9868	0.9868	0.7302	0.7302
CHI-PR 7 etiquetas	0.8832	0.8832	0.4879	0.4879	0.9943	0.9943	0.7281	0.7281
Full kNN	0.9428	0.9428	0.5087	0.5087	0.9996	0.9996	0.6934	0.6934

Tabla 3 Precisión en CHI-PA con limpieza aleatoria.

Los resultados que presenta CHI-PA en precisión son bastante buenos. En general está a la altura de CHI-PR y en algunos casos (Poker, Susy) le llega a superar en sus mejores configuraciones.

Los resultados que presenta en el dataset Skin en las configuraciones en las que se hace uso de la media dan pie a pensar que el uso de esta en algunos datasets pueda ser contraproducente a pesar de que no ocurra lo mismo en el resto de datasets. La decisión de si el uso de la media es beneficioso se puede decidir mejor tras ver los resultados en el ratio de reducción.

Los algoritmos de limpieza no han ayudado a la hora de aumentar la precisión, de hecho la bajan. El algoritmo de kNN reduce bastante la precisión, llegándola a bajar hasta 2 décimas. El algoritmo de limpieza aleatoria también llega a bajar la precisión pero no en tanta medida como en las versiones que utilizan kNN.

5.2 RESULTADOS DE CHI-PF EN PRECISIÓN

En este apartado se muestran los resultados en precisión que ha dado el algoritmo kNN usando los prototipos generados por CHI-PF en distintas configuraciones.

Los resultados están divididos en 3 tablas según método de limpieza que se ha usado, en primer lugar sin limpieza Tabla 4, luego con limpieza kNN tabla 5 y finalmente con limpieza aleatoria tabla 6.

Los resultados de cada configuración están ordenados por tablas distinguiéndose en cada una las versiones que han usado la media para filtrar las celdas de las que no. En primer lugar aparecen las distintas configuraciones de CHI-PF, luego diversas configuraciones de CHI-PR y finalmente los resultados que da el algoritmo kNN usando todo el dataset de entrenamiento original.

En cada columna se puede distinguir cual ha sido el mejor método, señalado en verde, y el peor señalado en rojo.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PF 80%	0.8603	0.8645	0.5432	0.5432	0.9907	0.9859	0.7302	0.7338
CHI-PF 90%	0.8557	0.8691	0.5393	0.5393	0.9907	0.9899	0.7318	0.7353
Mejor conf. CHI-PR	0.8087	0.8087	0.5249	0.5249	0.9893	0.9893	0.7215	0.7215
Mejores CHI-PR	0.8930	0.8930	0.5350	0.5350	0.9986	0.9986	0.7302	0.7302
CHI-PR 3 etiquetas	0.7749	0.7749	0.5334	0.5334	0.9753	0.9753	0.6877	0.6877
CHI-PR 5 etiquetas	0.8474	0.8474	0.5061	0.5061	0.9868	0.9868	0.7302	0.7302
CHI-PR 7 etiquetas	0.8832	0.8832	0.4879	0.4879	0.9943	0.9943	0.7281	0.7281
Full kNN	0.9428	0.9428	0.5087	0.5087	0.9996	0.9996	0.6934	0.6934

Tabla 4 Precisión en CHI-PF sin limpieza.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PF 80%	0.8486	0.8514	0.5432	0.5432	0.9904	0.9858	0.7143	0.7168
CHI-PF 90%	0.8408	0.8521	0.5393	0.5393	0.9897	0.9894	0.7123	0.7149
Mejor conf. CHI-PR	0.8087	0.8087	0.5249	0.5249	0.9893	0.9893	0.7215	0.7215
Mejores CHI-PR	0.8930	0.8930	0.5350	0.5350	0.9986	0.9986	0.7302	0.7302
CHI-PR 3 etiquetas	0.7749	0.7749	0.5334	0.5334	0.9753	0.9753	0.6877	0.6877
CHI-PR 5 etiquetas	0.8474	0.8474	0.5061	0.5061	0.9868	0.9868	0.7302	0.7302
CHI-PR 7 etiquetas	0.8832	0.8832	0.4879	0.4879	0.9943	0.9943	0.7281	0.7281
Full kNN	0.9428	0.9428	0.5087	0.5087	0.9996	0.9996	0.6934	0.6934

Tabla 5 Precisión en CHI-PF con limpieza kNN.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PF 80%	0.8590	0.8638	0.5432	0.5431	0.9907	0.9859	0.7300	0.7337
CHI-PF 90%	0.8545	0.8680	0.5394	0.5393	0.9905	0.9899	0.7314	0.7351
Mejor conf. CHI-PR	0.8087	0.8087	0.5249	0.5249	0.9893	0.9893	0.7215	0.7215
Mejores CHI-PR	0.8930	0.8930	0.5350	0.5350	0.9986	0.9986	0.7302	0.7302
CHI-PR 3 etiquetas	0.7749	0.7749	0.5334	0.5334	0.9753	0.9753	0.6877	0.6877
CHI-PR 5 etiquetas	0.8474	0.8474	0.5061	0.5061	0.9868	0.9868	0.7302	0.7302
CHI-PR 7 etiquetas	0.8832	0.8832	0.4879	0.4879	0.9943	0.9943	0.7281	0.7281
Full kNN	0.9428	0.9428	0.5087	0.5087	0.9996	0.9996	0.6934	0.6934

Tabla 6 Precisión en CHI-PF con limpieza aleatoria.

Los resultados de CHI-PF en cuanto a precisión son bastante buenos. Comparándolo con CHI-PA, CHI-PF es más estable ya que ninguno de sus resultados pierde en comparación con los de CHI-PR y los mejores resultados de ambos están a la par. Al contrario que CHI-PA que perdía en precisión en el dataset Skin cuando se usaba la media.

Al igual que en CHI-PA, los algoritmos de limpieza bajan la precisión en todas las configuraciones de CHI-PF y tendría que valorarse su uso tras comprobar los resultados que dan en el ratio de reducción.

5.3 Resultados de CHI-PA en ratio de reducción

En este apartado se muestran los resultados en ratio de reducción que han dado las distintas configuraciones de CHI-PA.

Los resultados esta divididos en 3 tablas según método de limpieza siendo sin limpieza Tabla 7, con limpieza kNN tabla 8 y con limpieza aleatoria tabla 9.

Los resultados de cada configuración están ordenados por tablas distinguiéndose en cada una las versiones que han usado la media para filtrar las celdas de las que no. En primer lugar aparecen las distintas configuraciones de CHI-PA y luego diversas configuraciones de CHI-PR.

En cada columna aparecen señaladas que configuraciones han tenido los mejores y peores resultados, estando en verde las mejores configuraciones y en rojo las peores.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PA 0%	92.97	87.11	88.26	88.26	99.97	99.83	92.81	91.27
CHI-PA 10%	94.78	87.11	88.26	88.26	99.97	99.83	93.59	92.14
CHI-PA 20%	92.97	87.11	88.26	88.26	99.97	99.83	94.74	93.40
CHI-PA 30%	93.02	87.17	88.33	88.33	99.97	99.83	96.06	94.88
Mejor conf. CHI-PR	97.58	97.58	96.32	96.32	99.98	99.98	99.15	99.15
Mejores CHI-PR	99.81	99.81	99.99	99.99	99.99	99.99	99.99	99.99
CHI-PR 3 etiquetas	98.64	98.64	88.11	88.11	99.99	99.99	99.97	99.97
CHI-PR 5 etiquetas	96.10	96.10	99.48	99.48	99.96	99.96	99.89	99.89
CHI-PR 7 etiquetas	92.16	92.16	99.97	99.97	97.67	97.67	93.54	93.54

Tabla 7 Ratio de reducción en CHI-PA sin limpieza.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PA 0%	96.62	94.53	88.26	88.26	99.99	99.95	97.21	96.65
CHI-PA 10%	96.62	94.52	88.26	88.26	99.99	99.95	97.88	97.39
CHI-PA 20%	96.62	94.54	88.26	88.26	99.99	99.95	98.57	98.17
CHI-PA 30%	96.65	94.57	88.33	88.33	99.99	99.95	99.09	98.78
Mejor conf. CHI-PR	97.58	97.58	96.32	96.32	99.98	99.98	99.15	99.15
Mejores CHI-PR	99.81	99.81	99.99	99.99	99.99	99.99	99.99	99.99
CHI-PR 3 etiquetas	98.64	98.64	88.11	88.11	99.99	99.99	99.97	99.97
CHI-PR 5 etiquetas	96.10	96.10	99.48	99.48	99.96	99.96	99.89	99.89
CHI-PR 7 etiquetas	92.16	92.16	99.97	99.97	97.67	97.67	93.54	93.54

Tabla 8 Ratio de reducción en CHI-PA con limpieza kNN.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PA 0%	93.57	88.19	88.26	88.26	99.98	99.86	93.10	91.59
CHI-PA 10%	93.57	88.19	88.26	88.26	99.98	99.86	93.89	92.46
CHI-PA 20%	93.57	88.19	88.26	88.26	99.98	99.86	95.02	93.70
CHI-PA 30%	93.62	88.25	88.33	88.33	99.98	99.86	96.30	95.13
Mejor conf. CHI-PR	97.58	97.58	96.32	96.32	99.98	99.98	99.15	99.15
Mejores CHI-PR	99.81	99.81	99.99	99.99	99.99	99.99	99.99	99.99
CHI-PR 3 etiquetas	98.64	98.64	88.11	88.11	99.99	99.99	99.97	99.97
CHI-PR 5 etiquetas	96.10	96.10	99.48	99.48	99.96	99.96	99.89	99.89
CHI-PR 7 etiquetas	92.16	92.16	99.97	99.97	97.67	97.67	93.54	93.54

Tabla 9 Ratio de reducción en CHI-PA con limpieza aleatoria.

Los resultados en el ratio de reducción no son tan buenas como los de precisión. El ratio de reducción no está a la altura de CHI-PR en sus mejores configuraciones y en algunos casos se queda bastante atrás. Evidentemente el peor resultado lo ha dado CHI-PA sin media ni limpieza en sus configuraciones menos restrictivas ya que es la que saca más prototipos, en consecuencia también es de las que saca mejores resultados de precisión.

Los resultados también indican que el uso de las medias y los porcentajes son beneficiosos a la hora de aumentar el ratio de reducción sin que llegue a afectar negativamente en la precisión.

Se ha comprobado que los algoritmos de limpieza ayudan en el ratio de reducción enormemente. El algoritmo de limpieza aleatoria consigue un leve aumento del ratio de reducción llegándose a compensar por la leve perdida que produce en los resultados de precisión. El algoritmo de limpieza kNN consigue unos resultados muy buenos, poniendo a CHI-PA a la altura de algunas de las versiones de CHI-PR haciendo razonable su uso a pesar de la perdida que provoca en los resultados de precisión.

5.4 RESULTADOS DE CHI-PF EN RATIO DE REDUCCIÓN

En este apartado se muestran los resultados en ratio de reducción que han dado las distintas configuraciones de CHI-PF.

Los resultados esta divididos en 3 tablas según método de limpieza siendo sin limpieza Tabla 10, con limpieza kNN tabla 11 y con limpieza aleatoria tabla 12.

Los resultados de cada configuración están ordenados por tablas distinguiéndose en cada una las versiones que han usado la media para filtrar las celdas de las que no. En primer lugar aparecen las distintas configuraciones de CHI-PF y luego diversas configuraciones de CHI-PR.

En cada columna aparecen señaladas que configuraciones han tenido los mejores y peores resultados, estando en verde las mejores configuraciones y en rojo las peores.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PF 80%	96.03	95.48	95.75	95.75	99.96	99.98	96.77	96.09
CHI-PF 90%	96.55	94.82	97.10	97.10	99.96	99.97	96.25	95.63
Mejor conf. CHI-PR	97.58	97.58	96.32	96.32	99.98	99.98	99.15	99.15
Mejores CHI-PR	99.81	99.81	99.99	99.99	99.99	99.99	99.99	99.99
CHI-PR 3 etiquetas	98.64	98.64	88.11	88.11	99.99	99.99	99.97	99.97
CHI-PR 5 etiquetas	96.10	96.10	99.48	99.48	99.96	99.96	99.89	99.89
CHI-PR 7 etiquetas	92.16	92.16	99.97	99.97	97.67	97.67	93.54	93.54

Tabla 10 Ratio de reducción en CHI-PF sin limpieza.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PF 80%	97.12	96.78	95.75	95.75	99.97	99.98	98.50	98.18
CHI-PF 90%	97.64	96.66	97.10	97.10	99.97	99.98	98.68	98.41
Mejor conf. CHI-PR	97.58	97.58	96.32	96.32	99.98	99.98	99.15	99.15
Mejores CHI-PR	99.81	99.81	99.99	99.99	99.99	99.99	99.99	99.99
CHI-PR 3 etiquetas	98.64	98.64	88.11	88.11	99.99	99.99	99.97	99.97
CHI-PR 5 etiquetas	96.10	96.10	99.48	99.48	99.96	99.96	99.89	99.89
CHI-PR 7 etiquetas	92.16	92.16	99.97	99.97	97.67	97.67	93.54	93.54

Tabla 11 Ratio de reducción en CHI-PF con limpieza kNN.

Dataset	Covtype		Poker		Skin		Susy	
Algoritmo	Con media	Sin media	Con media	Sin media	Con media	Sin media	Con media	Sin media
CHI-PF 80%	96.14	95.58	95.75	95.75	99.97	99.98	96.85	96.17
CHI-PF 90%	96.69	94.99	97.10	97.10	99.97	99.97	96.39	95.76
Mejor conf. CHI-PR	97.58	97.58	96.32	96.32	99.98	99.98	99.15	99.15
Mejores CHI-PR	99.81	99.81	99.99	99.99	99.99	99.99	99.99	99.99
CHI-PR 3 etiquetas	98.64	98.64	88.11	88.11	99.99	99.99	99.97	99.97
CHI-PR 5 etiquetas	96.10	96.10	99.48	99.48	99.96	99.96	99.89	99.89
CHI-PR 7 etiquetas	92.16	92.16	99.97	99.97	97.67	97.67	93.54	93.54

Tabla 12 Ratio de reducción en CHI-PF con limpieza aleatoria.

Al igual que en los resultados de precisión, CHI-PF consigue resultados más estables que CHI-PA sobrepasándolo en algunas configuraciones. El hecho de que CHI-PF filtre un dataset mediante varias iteraciones hace que sus resultados en ratio de precisión mejoren a los de CHI-PA. A pesar de que pueda parecer innecesario el uso de la limpieza en el algoritmo CHI-PF, los resultados indican que le ayudan a ganar aún más ratio de reducción a costa de perder precisión.

5.5 RESUMEN DE RESULTADOS

En este apartado se hará un análisis general de los resultados de cada método y de las configuraciones usadas para finalmente elegir un método que sea el mejor.

- **Porcentajes**: El uso de porcentajes en ambos métodos ha sido beneficioso. Ha permitido aumentar el ratio de reducción sin que la precisión se vea afectada. Dependiendo del dataset que se utilice la precisión varia, en cuanto a ratio de reducción, cuanto más restrictivo sea, mayor reducción se consigue. Con los datos obtenidos lo mejor es dejar una configuración intermedia.
- **Media**: El uso de la media ha dado unos resultados ambiguos. Al usarse en CHI-PA se ha reducido la precisión pero en CHI-PF no ha perdido. En ambos casos ha ayudado a mejorar el ratio de reducción.
- **Limpieza**: La limpieza ha ayudado enormemente a aumentar el ratio de reducción pero a costa de la precisión. La limpieza aleatoria no genera unos cambios suficientemente buenos como para que pueda ser tenida en cuenta debido a que poner más esfuerzos en métodos de filtrado ha resultado más beneficioso que este tipo de limpieza. La limpieza mediante kNN ha ayudado muchísimo a la hora de aumentar el ratio de reducción pero también ha bajado en consecuencia la precisión volviéndose más un algoritmo de reducción más que de limpieza. A pesar de esto sus resultados no han sido malos y futuras investigaciones en cuanto a su configuración podrían hacerlo más útil.
- **CHI-PA**: El mayor problema que presenta este algoritmo es que genera un excesivo número de prototipos. El uso de distintos filtros junto con algoritmo de limpieza han logrado que se estabilice su ratio de reducción sin que pierda precisión en exceso. Sin embargo sus resultados en ratio de reducción han sido pobres en respecto a CHI-PR y por lo general se ve superado por CHI-PF. En cuanto a precisión el algoritmo ha demostrado ser bastante bueno superando por lo general a CHI-PR.

- **CHI-PF**: Este algoritmo ha dado buenos resultados tanto en precisión como en ratio de reducción. En todas sus versiones está a la par que CHI-PR en precisión pero un poco por detrás en ratio de reducción.

Tras ver los resultados se puede afirmar que ambos algoritmos presentan configuraciones estables para todos los datasets sin que la configuración tenga que ser modificada.

De las dos versiones planteadas la que ha dado mejores resultados ha sido CHI-PF. CHI-PA no ha dado malos resultados pero no ha demostrado tener la estabilidad que ofrece CHI-PF el cual lo convierte en un algoritmo más fiable. CHI-PF es el algoritmo que da más seguridad para ser usado en la actualidad por otros algoritmos de clasificación como para ser objeto de futuras mejora o investigaciones.

De todas las configuraciones testeadas de CHI-PF la mejor de todas para ser usada en la actualidad es la versión $CHI - PF\ con\ media\ y$ sin $limpieza$. Esta versión es la que ofrece una mayor precisión y un ratio de reducción aceptable aunque no óptimo. Sin embargo, en el caso de que en el problema en el que se fuera a usar se prefiriese el ratio de reducción sobre la precisión la versión a elegir seria $CHI - PF\ con\ media\ y$ con $limpieza\ kNN$.

Capítulo 6 CONCLUSIONES Y LÍNEAS FUTURAS

En este proyecto se ha experimentado sobre cómo mejorar el algoritmo de CHI-PR y si se consiguen mejoras ejecutándolo varias veces usando distintas configuraciones.

6.1 CONCLUSIONES

1. El uso iterativo de CHI-PR para generar prototipos estabiliza el algoritmo creando una versión que funciona bien con todos los datos. Esta versión estable es tan buena como CHI-PR en sus mejores configuraciones
2. El uso de distintas configuraciones de filtrado ayudan a no generar demasiados prototipos sin que la precisión se vea afectada de manera sencilla. Ayudando en la labor de distintos algoritmos de limpieza que podrían usarse posteriormente.
3. A lo largo del algoritmo se generan prototipos teniendo siempre en cuenta que no se genere ruido, sin embargo, cuando todos los prototipos generados son agrupados algunos podrían llegar a verse como ruido o innecesarios. El uso de una fase de limpieza al final del algoritmo no ha llegado a ayudar a limpiar el ruido pero si lo hace a la hora de aumentar el ratio de reducción. Futuras versiones con configuraciones más testeadas podrían llegar a limpiar el ruido y mejorar más el ratio de reducción sin llegar a afectar a la precisión.

6.2 LÍNEAS FUTURAS

A pesar de que se han obtenido buenos resultados, el algoritmo aún puede mejorarse y obtener mejor precisión y ratio de reducción si se estudian las siguientes partes.

6.2.1 Mejora de los antecedentes

Los antecedentes son clave a la hora de agrupar correctamente las instancias. Dependiendo de cómo se agrupen se obtendrán unos prototipos u otros. En este proyecto se han usado un número fijo de etiquetas lingüísticas para generar antecedentes pero no todos los datasets se comportan igual bajo el mismo número de etiquetas lingüísticas. Por ello un aspecto a mejorar sería el de dejar la configuración estática de etiquetas lingüísticas por otra dinámica que se adapte a los datos de entrenamiento. Con ello mejoraría la calidad de los antecedentes y los prototipos generados aumentando así la precisión.

6.2.2 Limpieza de los prototipos

Como se ha podido comprobar, los mejores resultados en cuanto a precisión han venido de las versiones que han generado más prototipos. Partiendo de estas versiones se pueden implementar diversos algoritmos de limpieza para mejorar el ratio de reducción intentando siempre no perder precisión en exceso.

6.2.3 Cantidad y calidad de datasets

Uno de los problemas del BigData es que no solo aumenta la cantidad de los datos, también aumenta el tiempo necesario para procesarlos. Los datasets usados en este proyecto son algunos de los que utiliza CHI-PR para comprobar sus resultados. Se valoró la opción de hacer pruebas con otros datasets pero debido a la gran cantidad de tiempo que era necesario para realizar pruebas con estos, la idea se descartó.

Algunos de los datasets usados tienen una mayor cantidad de atributos nominales que numéricos haciendo que no fuesen los más idóneos para algoritmos con una base en lógica difusa tan fuerte.

En futuros proyectos sería muy interesante usar una mayor cantidad de datasets más idóneos para este tipo de algoritmo de generación de prototipos.

BIBLIOGRAFÍA

[1] M. Elkano, M. Galar, J. Sanz, H. Bustince: CHI-PR: A MapReduce Prototype Reduction algorithm with linear time complexity for BigData classification.

[2] Ishibuchi, H., Nakashima, T., Nii, M., 2004. Classification and modeling with linguistic information granules: Advanced approaches to linguistic Data Mining. Springer-Verlag.

[3] M. Elkano, M. Galar, J. Sanz, H. Bustince: A Fuzzy Rule-Based Classifier System for Big Data classification problems.

[4] H. Karau, A. Konwinski, P. Wendell, M. Zaharia: Learning Spark.

[5] C. Lam: Hadoop in action.

[6] J. Swartz: Learning Scala.

[7] A. Alexander: Scala Cookbook.