Doctorate Programme in Information and Knowledge Engineering

Programa de Doctorado en Ingeniería de la Información y del Conocimiento

On the Design of Distributed and Scalable Feature Selection Algorithms

Presented by

Raul Jose Palma Mendoza

Advisors:
Luis de Marcos Ortega, PhD
Daniel Rodriguez Garcia, PhD

Alcalá de Henares, 2019

Abstract

Feature selection is an important stage in the pre-processing of the data prior to the training of a data mining model or as part of many data analysis processes. The objective of feature selection consists in detecting within a group of features which are the most relevant and which are redundant according to some established metric. With this, it is possible to create more efficient and interpretable data mining models, also, by reducing the number of features, data collection costs can be reduced in future. Currently, according to the phenomenon widely known as "big data", the datasets available for analyze are growing in size. This causes that many existing algorithms for data mining become unable to process them completely and even, depending on their size, feature selection algorithms themselves, also become unable to process them directly. Considering that this trend towards the growth of datasets is not expected to cease, the existence of scalable feature selection algorithms that are capable of increasing their processing capacity taking advantage of the resources of computer clusters becomes very important.

The following doctoral dissertation presents the redesign of two widely known feature selection algorithms: ReliefF and CFS, both algorithms were designed with the purpose of being scalable and capable of processing large volumes of data. This is demonstrated by an extensive comparison of both proposals with their original versions, as well as with other scalable versions designed for similar purposes. All comparisons were made using large publicly available datasets. The implementations were made using the Apache Spark tool, which has noways become a reference framework in the "big data" field. The source code written has been made available through a GitHub public repository [1,2].

[1] `https://github.com/rauljosepalma/DiCFS`

[2] `https://github.com/rauljosepalma/DiReliefF`

Resumen

La selección de atributos es una importante etapa en el preprocesamiento de los datos previo al entrenamiento de un modelo en minería de datos o como parte de cualquier proceso de análisis de datos. El objetivo de la selección de atributos consiste detectar dentro de un grupo de atributos cuáles son los más relevantes y cuáles son redundantes de acuerdo a alguna métrica establecida. Con esto se logra crear modelos de minería de datos de forma más eficiente y fáciles de interpretar, también, al detectar atributos pocos relevantes se puede ahorrar costo en futuras recolecciones de datos. Sin embargo actualmente, de acuerdo al fenómeno ampliamente conocido como "big data", los conjuntos de datos que se desea analizar son cada vez mayores. Esto provoca que muchos algoritmos existentes para minería de datos sean incapaces de procesarlos completos e incluso, dependiendo de su tamaño, tampoco puedan ser procesados directamente por los mismos algoritmos de selección de atributos. Considerando que esta tendencia al crecimiento de los conjuntos de datos no se espera cesará, se vuelve necesaria la existencia de algoritmos de selección de atributos escalables que sean capaces de aumentar su capacidad de procesamiento aprovechando los recursos de clúster de computadoras.

La siguiente disertación doctoral presenta el rediseño de dos algoritmos de selección de atributos ampliamente utilizados: ReliefF y CFS, ambos algoritmos fueron rediseñados con el propósito de ser escalables y capaces del procesamiento de grandes volúmenes de datos. Esto queda demostrado mediante un extensiva comparación de ambas propuestas con sus versiones originales así como también con otras versiones escalables diseñadas para propósitos similares. Todas las comparaciones se realizaron usando grandes conjuntos de datos de acceso público. Las implementaciones se realizaron utilizando la herramienta Apache Spark, que actualmente se ha convertido en todo un referente en el área de "big data". El código fuente escrito se ha puesto disponible en un repositorio público de GitHub a nombre del autor[3,4].

[3] `https://github.com/rauljosepalma/DiCFS`

[4] `https://github.com/rauljosepalma/DiReliefF`

Dedication and acknowledgements

Definitely doing a doctoral thesis is a great challenge, and to achieve it, one requires much more than having technical skills in the subject of research and have a lot of motivation and desire to investigate. It is a challenge that requires a huge strength, to get up and try again and again without being discouraged, although the results did not seem to arrive, even if nothing is reaped after much sowing, sacrificing, among other things, time with the most loved ones.

This strength, in my case I did not find it within myself, I must admit that it came from many people who in one way or another added their strength to mine and for that reason I could finish this effort reflected in the document below. The first to add were my parents Elda Marina Mendoza and Raúl Ovidio Palma (RIP) who with their example of effort to get ahead as a family gave me the greatest impulse. Next to my parents is the rest of my family, my aunts and uncles: Alba, Lupe, Chago and Saúl and my grandmother Angelina who lived a much harder life than everyone else in the family, and gave us a much higher example of sacrifice and effort than the made to date.

The second in adding a lot was my new family, my wife Aneliza and our two children: Ane Sofía and Ian. How not to thank my wife for all her wait during the almost 12 months we were thousands of miles away, all her effort to cover my absences, all her understanding and all the words, calls and gestures that encouraged me to continue during these years. To Ane Sofía and Ian because by making me a father, they injected me with a new strength and motivation that could not arrive from any other way.

The third ones in adding were my thesis directors: Luis and Dani, how many times they encouraged me, they corrected me, they filled me with hope. They definitely made a great team in directing this process. Also added a lot those who helped me feel a little closer to home: Fernando Serrano my roommate, Sara and Javi, Carlos, Sandra, Alicia, Ana and the priests of the parish Santo Tomas de Villanueva: Don Javier, Teo, Alberto and Luis Enrique how much courage did they inject me and how many times did they cure my soul with the gifts that God has given them. Since, it is not enough with a mental or physical strength, it was also necessary a lot of spiritual strength that only come from God who is the one who is behind all of this, from the miraculous approval of the scholarship to the most miraculous culmination of this project despite the difficulties.

I received the last impulse needed to complete this process during my research stay with the LIDIA group at the University of A Coruña, thanks to Amparo, Carlos, Isaac, Verónica and Laura for the excellent reception, for the opportunity of collaboration and for making me feel part of the team.

Finally, I also want to thank my co-workers at the UNAH, Servio for motivating me to participate in the scholarship call, to my colleagues in the laboratory “from the back of the hall” in Alcalá: Ana, Juan, Javi, Kike and Nancy. The current dean of the faculty of engineering of the

UNAH: Eduardo Gross and the previous dean Eng. Mónico Oyuela and in a special way to Daniel Meziat for his help with the initial procedures and for being aware of me. In general, I thank Fundación Carolina and the UNAH for the economic support and job stability that were key to completing this project.

Dedicatoria y agradecimientos

Definitivamente realizar un tesis doctoral es un gran reto, y para lograrlo se requiere mucho más que contar con competencias técnicas en el tema de desarrollar y tener amplia motivación y deseo de investigar. Es un reto que requiere de una fortaleza enorme, para levantarse e intentar una y otra vez sin desanimarse, aunque los resultados no parezcan llegar, aunque no se coseche nada después de mucho sembrar sacrificando de entre otras cosas, el tiempo con los seres más amados.

Esta fortaleza, en mi caso no la encontré dentro de mí mismo, debo reconocer que provino de muchas personas que de una u otra forma sumaron sus fuerzas a la mía y por esa razón pude culminar este esfuerzo reflejado en el documento a continuación. Los primeros en sumar fueron mis padres Elda Marina Mendoza y Raúl Ovidio Palma (QDDG) quienes con su ejemplo de esfuerzo por salir adelante como familia me dieron el más grande de los impulsos. Junto a mis padres está toda mi familia, mis tías y tíos: Alba, Lupe, Chago y Saúl y mi abuelita Angelina que con una vida mucho más difícil y dura que la de todos, nos dio un ejemplo mucho más alto de sacrificio y esfuerzo que el realizado hasta la fecha.

La segunda en sumar mucho fue mi nueva familia, mi esposa Aneliza y nuestros dos hijos: Ane Sofía e Ian. A mi esposa cómo no agradecerle toda su espera durante los casi 12 meses que estuvimos a miles kilómetros de distancia, todo su esfuerzo por cubrir mis ausencias, toda su comprensión y todas las palabras, llamadas y gestos que me animaron a seguir durante estos años. A Ane Sofía e Ian porque al hacerme un papá, me inyectaron una nueva fuerza y motivación que de otro lado no podía surgir.

Los terceros en sumar fueron mis directores de tesis: Luis y Dani, cuántas veces me animaron, me corregieron, me llenaron de esperanza, definitivamente hicieron una gran equipo al dirigir este proceso. También sumaron mucho aquellos que me ayudaron a sentir un poco más cerca de casa: Fernando Serrano mi compañero de piso, Sara y Javi, Carlos, Sandra, Alicia, Ana y los sacerdotes de la parroquia Santo Tomás de Villanueva: Don Javier, Teo, Alberto y Luis Enrique cuánto ánimo me inyectaron y cuántas veces me curaron el alma con los dones que Dios les ha dado. Pues no basta con una fortaleza mental ni física, fue necesaria también mucha fortaleza espiritual que sólo vino de Dios, quién es al final el que está detrás de todo esto desde la milagrosa aprobación de la beca de estudios hasta la más milagrosa culminación de este proyecto a pesar de las dificultades.

El último impulso que necesitaba para culminar este proceso lo recibí en mi estancia de investigación con el grupo LIDIA en la Universidad de A Coruña, gracias a Amparo, Carlos, Isaac, Verónica y Laura por la excelente acogida, por la oportunidad de colaboración y por hacerme sentir parte del equipo.

Finalmente, quiero agradecer también a mis compañeros de trabajo de la UNAH, a Servio por motivarme a participar en la convocatoria de becas, a mis compañeros en el laboratorio "del fondo del pasillo" en Alcalá: Ana, Juan, Javi, Kike y Nancy. Al decano actual de la facultad de Ingeniería

de la UNAH Eduardo Gross y al anterior decano Ing. Mónico Oyuela y de forma especial a Daniel Meziat por su ayuda con los trámites iniciales y por estar pendiente de mí. De forma general, doy gracias a la Fundación Carolina y la UNAH por el apoyo económico y la estabilidad laboral que fueron claves para poder culminar este proyecto.

Resumen Extendido

En los últimos años, un fenómeno conocido como *big data* ha sido reconocido en los campos académico e industrial. Esencialmente, el big data se refiere a la creciente cantidad de datos que está produciendo la sociedad de la información actual en prácticamente todas las áreas del conocimiento. Junto con el big data, han surgido desafíos sin precedentes para los científicos, ingenieros y profesionales que trabajan con datos y pretenden aprovechar su valor. El aumento exponencial en la cantidad de datos que están disponibles para ellos, hace que la tarea de procesar y analizar estos datos sea compleja y altamente exigente de recursos computacionales.

Para generar valor a partir de los datos se debe seguir un proceso. El proceso de descubrimiento de conocimiento en bases de datos (proceso KDD por sus siglas en inglés) [51] es un marco general que indica los pasos que se deben seguir para obtener conocimiento valioso de un conjunto de datos. El paso central en el proceso KDD se conoce como *minería de datos* en cual se utilizan técnicas especiales para crear un modelo que extrae patrones útiles y valiosos (conocimiento) de los datos. Otro paso importante del proceso de KDD es el *preprocesamiento de datos*, éste es un paso preparatorio pero notable, que si no se realiza con cuidado, puede hacer imposible obtener conocimiento valioso a partir de los datos. Además, el preprocesamiento de datos es un paso general que involucra muchas técnicas que pueden aplicarse a los datos originales, una de esas técnicas se conoce como *selección de atributos*.

La selección de atributos, en un sentido amplio, es una técnica de preprocesamiento de datos que se utiliza para reducir la cantidad de atributos que tiene un conjunto de datos. En términos simples, si se considera que un conjunto de datos está formado por un grupo de instancias, pueden ser: correos electrónicos, pacientes, intentos de conexión, perfiles de usuario, imágenes, etc. Los atributos son las propiedades o características almacenadas para cada instancia, por ejemplo, para un correo electrónico, los atributos pueden ser: asunto, fecha de envío, remitente, destinatario, contenido, etc.

Según Guyon and Elisseeff [66], las técnicas de selección de atributos se aplican para lograr al menos uno de los siguientes objetivos:

- Mejorar la calidad de los resultados del modelo producido.
- Hacer que la creación (entrenamiento) de un modelo sea más eficiente en términos de consumo de recursos computacionales.

- Mejorar los modelos resultantes haciéndolos más pequeños y más fáciles de entender.

La era actual de big data trae consigo la aparición frecuente de conjuntos de datos con alta dimensionalidad, es decir, conjuntos de datos con un gran número de atributos que, para muchas de las técnicas actuales de minería de datos, pueden causar el efecto conocido como *maldición de la dimensionalidad* [11] que se refiere al hecho de que el número de pasos para crear un modelo que debe seguir una técnica específica crece demasiado rápido con el número de atributos y la probabilidad de obtener un modelo no válido o ningún modelo puede volverse demasiado alta cuando hay muchos atributos presentes.

En este contexto, la selección de atributos se convierte en un paso extremadamente importante dentro del preprocesamiento de datos [60], convirtiéndose en algunos casos en la única forma de producir resultados valiosos especialmente para aquellas técnicas de minería de datos que son más sensibles a la maldición de la dimensionalidad. Sin embargo, los conjuntos de datos de alta dimensión que aparecen hoy en día con más frecuencia no solo pueden causar problemas a las técnicas de minería de datos sino también a las técnicas tradicionales de selección de atributos, esto es especialmente cierto para los algoritmos de selección de atributos multivariable, los cuales son de alta importancia ya que tienen la capacidad para considerar las dependencias de las atributos en sus resultados, una propiedad deseable al aplicar una técnica de selección de atributos. Además, las técnicas de selección de atributos (y la minería de datos en general) no solo pueden verse afectadas por la cantidad de atributos que tiene un conjunto de datos, sino también por la cantidad de instancias (filas). De manera similar, los problemas que surgen con al aumento en el número de instancias están relacionados con un aumento en los recursos computacionales que exige el algoritmo. En algunos casos, esta demanda excede los recursos disponibles y evita la ejecución del algoritmo. Además, a menudo es conveniente considerar todas las instancias disponibles, especialmente en problemas complejos, ya que es bien sabido que tener más instancias puede mejorar la calidad de los modelos resultantes [68]. Considerando todo esto, Bolón-Canedo et al. [13] declaran que “existe una evidente necesidad de adaptar los métodos de selección de atributos existentes o proponer nuevos para enfrentar los desafíos planteados por la explosión de big data”, y esto de hecho se convierte en la principal motivación de la actual investigación.

Objetivo y Metodología de Investigación

En las últimas décadas, se han desarrollado posiblemente cientos de métodos de selección de atributos, algunos de ellos han sobresalido sobre el resto, se han considerado en varias revisiones bibliográficas [15, 25, 138] y, por supuesto, se han utilizado en muchos estudios aplicados. La declaración realizada por Bolón-Canedo et al. [13] y mencionada en la sección anterior ofrece un vistazo a dos vías de investigación: (i) desarrollo de nuevos métodos de investigación y (ii) mejora

de los métodos existentes. Esta tesis está dedicada a la última vía y por tanto, el objetivo general de esta tesis se puede enunciar de la siguiente manera:

> Desarrollar nuevas versiones de métodos existentes de selección de atributos ampliamente usados para que puedan hacer frente a grandes conjuntos de datos de forma escalable.

En este punto, es importante definir qué es un conjunto de datos grande. El Grupo de Investigación de Computación sobre Soft-Computing y Sistemas de Información Inteligentes de la Universidad de Granada tiene un repositorio de datos publicado [5] que incluye conjuntos de datos de fuentes ampliamente conocidas como ser el repositorio del conjunto de datos de Aprendizaje Automático de la Universidad de California Irvine [6] y otros provenientes de concursos académicos de procesamiento de datos a gran escala, este repositorio se utiliza como la principal fuente de datos públicos en esta tesis. La mayoría de los conjuntos de datos a los que se hace referencia allí tienen un número de instancias del orden de 10^6 y un número de atributos en el orden de 10^1 hasta 10^3.

Para lograr el objetivo declarado, la investigación realizada en este trabajo siguió los siguientes pasos:

1. Revisar las tecnologías más importantes para procesar y analizar grandes cantidades de datos.

2. Revisar toda la investigación accesible dedicada a la escalabilidad de los algoritmos de selección de atributos.

3. Identificar algunas de las técnicas de selección de atributos más relevantes y analizar cada una para determinar cuáles eran más propensas a ser rediseñadas de manera escalable. Después de realizar este paso, se seleccionaron dos técnicas de selección de atributos: ReliefF [88, 135] y CFS [70, 71], las principales razones de esta selección fueron: son algoritmos ampliamente utilizados con muchas aplicaciones, sus versiones actuales no se escalan bien con cantidades crecientes de datos, se encontró muy poca investigación con el objetivo de crear versiones escalables de ellas y las tecnologías descritas en el siguiente párrafo se evaluaron como aplicables para su rediseño e implementación.

4. Seleccionar un grupo de tecnologías para utilizarlas como plataforma de diseño e investigación para este trabajo, siguiendo criterios comunes como: apertura del código fuente, novedad, popularidad, buenos resultados en investigaciones anteriores, buen soporte, accesible para la investigación. Después de realizar este paso, la plataforma seleccionada fue: Apache Spark [162] para el procesamiento de grandes conjuntos de datos y Hadoop HDFS [19] para el almacenamiento distribuido de los conjuntos de datos.

[5] `https://sci2s.ugr.es/BigData#Datasets`
[6] `https://archive.ics.uci.edu/ml/datasets.html`

5. Diseñar e implementar las técnicas seleccionadas utilizando la plataforma de software elegida.

6. Probar, experimentar y comparar las versiones rediseñadas con las versiones originales utilizando grandes conjuntos de datos para determinar si realmente son escalables y más apropiadas para estas cantidades de datos.

Contribuciones y Publicaciones

Las contribuciones hechas en esta tesis deben quedar claras después de leer los pasos de investigación descritos en la sección anterior. Específicamente, los principales resultados de este trabajo son las versiones rediseñadas de dos técnicas de selección de atributos tradicionales y relevantes: ReliefF y CFS. De hecho, se realizaron dos publicaciones en revistas indexadas en JCR (Journal Citation Reports), una para cada versión rediseñada, que se enumeran a continuación junto con sus factores de impacto correspondientes en el momento de la redacción.

- Palma-Mendoza, R. J., Rodriguez, D., & de-Marcos, L. (2018). Distributed ReliefF-based feature selection in Spark. Knowledge and Information Systems, 1–20. `https://doi.org/10.1007/s10115-017-1145-y` (2.247 Impact Factor Second Quartile).

- Palma-Mendoza, R.-J., de-Marcos, L., Rodriguez, D., & Alonso-Betanzos, A. (2018). Distributed correlation-based feature selection in Spark. Information Sciences. `https://doi.org/10.1016/j.ins.2018.10.052` (4.305 Impact Factor First Quartile).

Con respecto a la primera contribución, el algoritmo ReliefF fue publicado por Kononenko [88] como una extensión de Relief [86, 87]. Dado que Relief solo era capaz de lidiar con problemas de clasificación binaria, ReliefF extendió sus capacidades para lidiar con problemas ruidosos, incompletos y de múltiples clases. Relief es reconocido como uno de los algoritmos de selección de atributos tipo filtro más destacados y ha dado origen a toda una familia de algoritmos, a veces conocidos como algoritmos basados en Relief o RBA por sus siglas en inglés [152], siendo ReliefF probablemente el más popular. Sin embargo, la complejidad computacional de ReliefF es $\mathcal{O}(m \cdot n \cdot a)$, donde n es el número de instancias en el conjunto de datos, m es el número de muestras tomadas de n instancias y a es el número de atributos. Por lo tanto, si el algoritmo se va a ejecutar considerando todas las instancias en el conjunto de datos, entonces $m = n$, y la función de complejidad crece de forma cuadrática con la cantidad de instancias $\mathcal{O}(n^2 \cdot a)$. Esta complejidad, junto con el hecho de que la mayoría de las implementaciones tradicionales necesitan cargar todo el conjunto de datos en la memoria para procesarlo, hace que las implementaciones tradicionales sean inutilizables con los grandes conjuntos de datos. La primera contribución de este trabajo consistió en el diseño e implementación de una nueva versión escalable del algoritmo original de ReliefF llamado DiReliefF. Esta nueva versión es capaz de aprovechar los recursos

computacionales de un clúster de computadoras para manejar grandes conjuntos de datos y brinda los mismos resultados que ReliefF habría arrojado si pudiera ejecutarse en los datos. De modo que, DiReliefF mantiene todas las propiedades ya estudiadas de ReliefF que lo han hecho popular entre los investigadores y los profesionales.

Con respecto a la segunda contribución, el algoritmo CFS (Correlation-based Feature Selection) fue publicado por Hall [70, 71]. CFS ha sido considerado en muchas ocasiones como una de las técnicas más importantes y ampliamente utilizadas en la selección de atributos. [13, 15, 96, 140]. Además, su creador Mark Hall, es también uno de los principales contribuyentes del software de minería de datos WEKA [69], una de las herramientas de minería de datos de código abierto y libre acceso más ampliamente utilizadas en el mundo. Por supuesto, CFS está incluido en WEKA junto con ReliefF entre otros. Sin embargo, al igual que ReliefF, el algoritmo CFS tiene problemas de escalabilidad, su complejidad computacional es $\mathcal{O}(a^2 \cdot m)$, donde a es el número de atributos y m es el número de instancias. Esta complejidad cuadrática en el número de atributos hace que CFS sea muy sensible a la maldición de la dimensionalidad. Por otro lado, la implementación de WEKA también requiere que el conjunto de datos se cargue en la memoria para procesarlo, descartando la posibilidad de ejecutarlo en conjuntos de datos más grandes. Es por esto que, la segunda contribución de este trabajo es una versión escalable de CFS llamada DiCFS. De nuevo, esta nueva versión fue diseñada para aprovechar un clúster de computadoras con el fin de manejar grandes conjuntos de datos y brinda los mismos resultados que CFS habría arrojado si se pudiera ejecutar en los datos. Una vez más, manteniendo todas las propiedades y beneficios del CFS que lo han convertido en una técnica de selección de atributos relevante y ampliamente utilizada.

Conclusiones

De forma resumida, después de realizar el proceso de diseño, experimentación y prueba de los algoritmos propuestos, fue posible concluir lo siguiente:

El algoritmo DiRelief se comparó con una versión no distribuida del algoritmo implementado en la plataforma WEKA. Los resultados mostraron que la versión no distribuida es poco escalable, es decir, no puede manejar grandes conjuntos de datos debido a los requisitos de memoria. Por el contrario, DiReliefF es completamente escalable y proporciona mejores tiempos de ejecución y uso de memoria cuando se trata de conjuntos de datos muy grandes. Los experimentos también mostraron que el algoritmo es capaz de devolver resultados estables con tamaños de muestra que son mucho más pequeños que el tamaño del conjunto de datos completo.

Con respecto al algoritmo DiCFS, se diseñaron e implementaron dos versiones DiCFS-hp y DiCFS-vp. Estas dos versiones esencialmente difieren en cómo se distribuye el conjunto de datos a través de los nodos del clúster. La primera versión distribuye los datos mediante la división de filas (instancias) y la segunda versión, basada en el trabajo de Ramírez-Gallego et al. [132],

distribuye los datos dividiendo las columnas (atributos). Como resultado de una comparación de cuatro vías entre DiCFS-vp y DiCFS-hp, una implementación no distribuida en WEKA y una versión distribuida para regresión [47], se pudo concluir lo siguiente:

- Tanto DiCFS-vp como DiCFS-hp pudieron manejar conjuntos de datos más grandes de una manera mucho más eficiente que la implementación clásica de WEKA. Además, en muchos casos, fueron la única forma viable de procesar ciertos tipos de conjuntos de datos debido a los requisitos prohibitivos de memoria de WEKA.
- Entre los esquemas de partición horizontal y vertical, la versión horizontal (DiCFS-hp) demostró ser la mejor opción en el caso general debido a su mejor escalabilidad y su modo de partición natural que permite al motor de Spark hacer un mejor uso de los recursos del clúster.
- Para problemas de clasificación, los beneficios obtenidos con DiCFS en comparación con la versión sin distribución pueden considerarse iguales o incluso mejores que los beneficios ya demostrados para el problema de regresión [47].

De forma general, es posible concluir que el objetivo de “desarrollar nuevas versiones de métodos existentes de selección de atributos ampliamente usados para que puedan hacer frente a grandes conjuntos de datos de forma escalable” se logró con éxito para los casos específicos de los algoritmos de selección de atributos de ReliefF y CFS . Ambas versiones están listas para servir como herramientas valiosas para otros investigadores y profesionales en diferentes campos que necesiten procesar grandes conjuntos de datos para sus propios objetivos.

Table of Contents

List of Tables

List of Figures

Part I

Background

CHAPTER 1

INTRODUCTION

In the last years, a phenomenon known as *big data* has been recognized at the academic and industrial fields. Essentially, big data refers to the increasing amount of data that is being produced by the current information society in practically all areas of knowledge. Together with big data, unprecedented challenges have emerged for scientists, engineers and practitioners that work with data and intend to leverage its value. The exponential increase in the amount of data that is available to them, causes that the task of processing and analyzing data has turn to be complex and highly demanding of computational resources.

In order to produce value from data, a process must be followed. The Knowledge Discovery in Databases process (KDD process) [51] is a general framework that marks the steps that must be followed to obtain valuable knowledge from a set of data. The core step in the KDD process is known as *data mining*, basically in this step, special techniques are used to create a model that extracts useful and valuable patterns (knowledge) from data. Another important step of the KDD process is *data preprocessing*, a preparatory but remarkable step that if not done with care can make it impossible to obtain valuable knowledge from data. Moreover, data preprocessing is a general step that involves many techniques that can be applied to the original data, one of such techniques is known as *feature selection*.

Feature selection, in a broad sense is a data preprocessing technique used in order to reduce the amount of features a dataset has. In simple terms, if a dataset is considered to be made of a group of instances be they: emails, patients, connection attempts, user profiles, pictures, etc. Features are simply the attributes stored for each instance, for example for an email its features could be: subject, date sent, sender, receiver, content, etc. According to Guyon and Elisseeff [66], feature selection techniques are applied pursuing at least one of the following objectives:

- Improve the quality of the results of the produced model.

- Make the creation (training) of a model faster or more cost-effective in terms of computational resource consumption.

- Improve the resulting models by making them smaller and easier to understand.

The current era of big data brings with it the frequent appearance of datasets with high dimensionality, i.e., datasets with a high number of features, that for many of the current data mining techniques can cause the effect known as the *curse of dimensionality* [11]. This refers to the fact that the number of steps that a specific technique must follow in order to create a model grows too fast with the number of features and the probability of obtaining an invalid model or no model at all can get overly high when too much features are present.

In this context, feature selection turns to be an extremely important data preprocessing step [60], becoming in some cases the only way of producing valuable results specially for those data mining techniques that are more sensible to the curse of dimensionality. However, high dimensional datasets that are nowadays appearing more frequently can not only cause problems to the data mining techniques but also to some feature selection techniques themselves, this is specially true for multivariate feature selection algorithms, which are important due to their ability to consider feature dependencies, a desirable property when applying a feature selection technique. Moreover, feature selection techniques (and data mining in general) can not only be affected by the number of features a dataset has but also by the number of instances (rows). Similarly, the problems that arise when the number of instances grows are related to an increase in the computational resources that the algorithm demands. In some cases, this demand exceeds the resources available and prevents the execution of the algorithm. Furthermore, is often desirable to consider all the available instances, specially in complex problems, since its well known that having more instances can improve the quality of the resulting models [68]. Considering all this, Bolón-Canedo et al. [13] state that "there is an evident need to adapt existing feature selection methods or propose new ones in order to cope with the challenges posed by the explosion of big data", and this in fact becomes the main motivation of the current investigation.

1.1 Research Objective and Methodology

In the last decades maybe hundreds of feature selection methods have been developed, some of them have excelled over the rest, have been considered in many literature reviews [15, 25, 138] and of course, have been used in many applied studies. The statement made by Bolón-Canedo et al. [13] and mentioned in the previous section gives a glimpse over two research paths: (i) developing new research methods and (ii) improving existing methods, this thesis is devoted to the latter path. Therefore, the general objective of this thesis can be enunciated as follows:

Develop new versions of existing important feature selection methods so that they are able to cope with large datasets in a scalable fashion.

At this point, it is important to define what a large dataset is. The Soft Computing and Intelligent Information Systems Research Group from the University of Granada has a published large dataset repository [1] that includes data from the well known University of California Irvine Machine Learning dataset repository [2] and other large scale data processing academic contests, this repository is used as the main source of public data in this thesis. Most of the datasets referenced in there have a number of instances in the order of 10^6 and number of features in the range of 28 until 2000.

In order to accomplish the stated objective, the research conducted in this work went through the following steps:

1. Review the most prominent technologies to process and analyze large amounts of data.

2. Review all the accessible research devoted to the scalability of feature selection algorithms.

3. Identify some of the most relevant feature selection techniques and analyze each one in order to determine which were more prone to be redesigned in a scalable manner. After performing this step, two feature selection techniques were selected: ReliefF [88, 135] and CFS [70, 71], the main reasons for this selection were: they are widely used algorithms with many applications, their current versions do not scale well with increasing amounts of data, very little research work was found with the aim of creating scalable versions of them and the technologies described in the next paragraph were applicable for their redesign and implementation.

4. Select a group of technologies in order to be used as the design and research platform for this work, following common criteria such as: source code openness, novelty, popularity, good results in previous research, good support, accessible for the research. After performing this step, the selected platform was: Apache Spark [162] for large dataset processing and Hadoop HDFS [19] for distributed storage of the datasets.

5. Design and implement the selected techniques using the chosen software platform.

6. Test, experiment and compare redesigned versions with the original versions using large datasets in order to determine if they were indeed scalable and more appropriate for these amounts of data.

[1] `https://sci2s.ugr.es/BigData#Datasets`
[2] `https://archive.ics.uci.edu/ml/datasets.html`

1.2 Contributions and Publications

The contributions made in this thesis should be clear after reading the research steps described in the previous section. Specifically, the main results of this work are the redesigned versions of two traditional and relevant feature selection techniques: ReliefF and CFS. In fact, two publications in JCR (Journal Citation Reports) indexed journals were made, one for each redesigned version, they are listed next together with their corresponding impact factors at the time of writing.

- Palma-Mendoza, R. J., Rodriguez, D., & de-Marcos, L. (2018). Distributed ReliefF-based feature selection in Spark. Knowledge and Information Systems, 1–20. `https://doi.org/10.1007/s10115-017-1145-y` (2.247 Impact Factor Second Quartile).

- Palma-Mendoza, R.-J., de-Marcos, L., Rodriguez, D., & Alonso-Betanzos, A. (2018). Distributed correlation-based feature selection in Spark. Information Sciences. `https://doi.org/10.1016/j.ins.2018.10.052` (4.305 Impact Factor First Quartile).

Regarding the first contribution, the ReliefF algorithm was published by Kononenko [88] as an extension of Relief [86, 87]. Since Relief was only capable of dealing with binary classification problems, ReliefF extended its capabilities to deal with noisy, incomplete, and multi-class problems. Relief is recognized as one of the most prominent filter-base feature selection technique and has given birth to whole family of algorithms sometimes known as a Relief-based algorithms (RBA) [152], being ReliefF probably the most popular. However, ReliefF's computational complexity is $\mathcal{O}(m \cdot n \cdot a)$, where n is the number of instances in the dataset, m is the number of samples taken from the n instances and a is the number of features. Thus, if the algorithm is to be executed considering all the instances in the dataset, then $m = n$, and the complexity function grows quadratically with the number of instances $\mathcal{O}(n^2 \cdot a)$. This complexity together with the fact that most of the traditional implementations need to load the whole dataset in memory in order to process it, turns traditional implementations unusable with large datasets. The first contribution of this work consists in the design and implementation of a new scalable version of the original ReliefF algorithm named DiReliefF. This new version is able to leverage the computational resources of a cluster of computers in order to handle large datasets providing the same results that ReliefF would have returned if it could be executed on the data. There by, DiReliefF maintains all the already studied properties of ReliefF that have made it popular between researchers and practitioners.

Respecting the second contribution, the CFS (Correlation-Based Feature Selection) algorithm was published by Hall [70, 71]. CFS has been considered in many occasions [13, 15, 96, 140] as one of the most important and widely used techniques in feature selection. Moreover, its creator Mark Hall, is also one of the main contributors of the WEKA data mining software [69] one of the most widely used open source and freely available data mining tools in the world and, of course, the CFS algorithm is included in WEKA together with ReliefF among others. However, similarly to

ReliefF, the CFS algorithm has scalability issues, its computational complexity is $\mathcal{O}(a^2 \cdot m)$, where a is the number of features and m is the number of instances. This quadratic complexity in the number of features makes CFS very sensitive to the *the curse of dimensionality* [10]. On the other hand, the WEKA implementation also requires the dataset to be loaded in memory to process it, ruling out the possibility of executing it in larger datasets. Thus, the second contribution of this work is a redesigned scalable version of CFS named DiCFS. This new version was again designed to leverage a computer cluster in order to handle large datasets providing the same results that CFS would have returned if it could be executed on the data. This new version also maintains all the properties and benefits of CFS that have made it a relevant and widely used feature selection technique.

1.3 Overview of the document

This dissertation is organized in three parts. Part I begins with this introductory chapter and then presents all the background concepts that support the contributions made using three chapters. Chapter 2, is devoted to the main topic of this work: *feature selection*, first establishing its importance and relation to the *machine learning* and *data mining* fields and then presenting a classification of current methods and evaluation metrics. Chapter 3 is titled *Big Data and Other Related Terms*, is a vital chapter to understand the necessity of having scalable algorithms to process the increasing amounts of data becoming available nowadays, it also presents and tries to establish relations of many other terms that have aroused somewhat together with *big data*, such as *data science, business intelligence and analytics*. Next, Chapter 4 discusses in a practical manner, the theory of *distributed systems* quickly turning to the three main technologies that conform the framework where the algorithms in this work were implemented, namely *MapReduce, Apache Hadoop* and *Apache Spark*. Part I ends with Chapter 5, that establishes the link between the first and second part of this work by presenting the last background concept: *distributed feature selection* and then reviewing the related work in the field paying special attention to the two algorithms redesigned in this thesis: ReliefF and CFS.

Part II details the contributions of this dissertation using a chapter for each algorithm: Chapter 6 for *DiReliefF* and Chapter 7 for *DiCFS*, both chapters include all the experiments, comparisons and results obtained with the proposed versions.

Finally, Part III (Chapter 8) concludes the dissertation and discusses future work.

CHAPTER 2

FEATURE SELECTION

2.1 Knowledge Discovery in Databases Process

In order to adequately contextualize the topic addressed in this thesis, it is imperative to place the field of *feature selection* on its place, for which it is valuable to start this discussion with the following topics: *Knowledge Discovery in Databases Process or KDD process*, *data mining* and *machine learning* as they constitute the environment within which the algorithms presented here take participation and special relevance.

The need to develop new methods and techniques in order to analyze the data automatically or semi-automatically has several decades being enunciated in the literature, and has gone hand in hand with the sustained growth in the storage, transmission rates and data processing the computers have had. Fayyad et al. [51] present the KDD process as a consequence of this need and consider it an attempt to address the problem of data overload that the era of digital information brought with it.

Fayyad et al. [51] define the KDD process as a non-trivial process to identify valid, novel, potentially useful and understandable patterns in the data. This being a process, has a series of stages that allow reaching its final objective which, in summary, consists of obtaining knowledge from the data. Figure 2.1 shows us these stages and gives us an indication of the iterative and interactive nature of this process, which refers to the fact that although there is a main flow between each of the stages, it is also possible that there are cycles between any of them. A briefly description of each one based on García et al. [57] is given below.

- *Understanding and specifying the problem*. This stage involves the understanding of the domain of the problem, the clear identification of the objective pursued with the KDD process and the selection of the data that will be used.

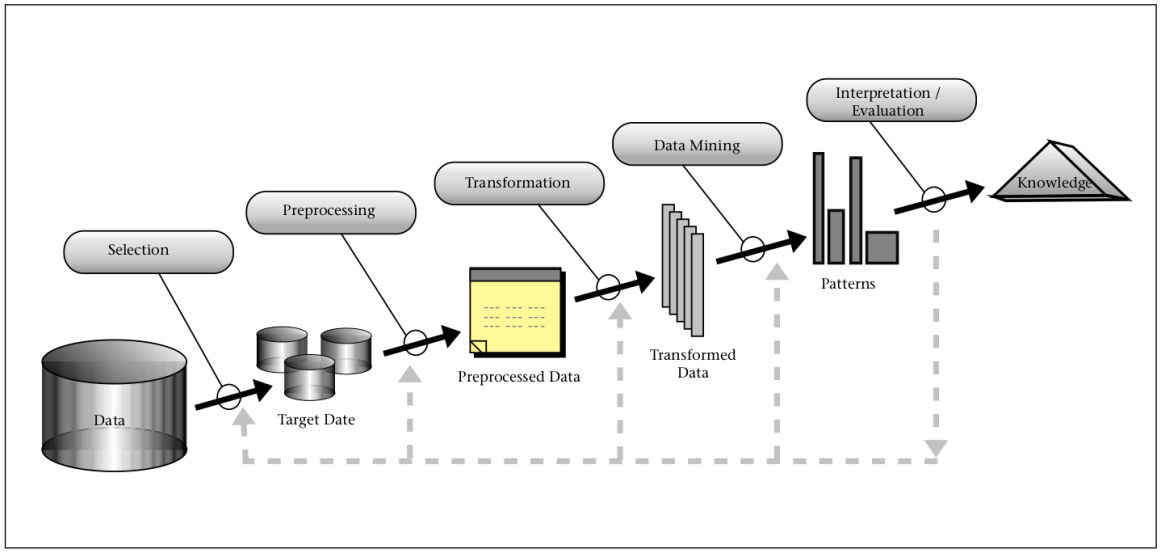

Figure 2.1: KDD process stages [51]

- *Data preprocessing*. This includes the cleaning of the data, the integration of the data when it is obtained from various sources, the transformation of the data in ways that may make it more useful for the next stage and the reduction of the data by eliminating instances (rows) or features (columns) of the dataset.
- *Data Mining*. It is the central point of the process, where different methods can be applied to extract valid and interesting patterns from the prepared data. This stage involves selecting the most suitable mining method for its adjustment and validation.
- *Evaluation*. In this stage, the patterns obtained are estimated and interpreted according to their interest and the objective identified at the beginning.
- *Exploitation of results*. Finally, the knowledge obtained can be used directly by incorporating it into another system or it can simply be reported using perhaps visualization tools.

2.2 Data Mining and Machine Learning

As mentioned in the previous section, data mining is the central step of the KDD process. According to Witten et al. [158], data mining is the process through which patterns, structures and theories are discovered in the data. This process is carried out automatically or semi-automatically and the information found after being evaluated and interpreted allows obtaining knowledge that has a scientific or economic value. In addition, the information has the characteristic of being hidden or at least not detectable by the naked eye, so to reveal it, data mining uses techniques that come from different areas of knowledge such as *statistics and probability*, *theory of databases* and machine learning especially.

In what corresponds to machine learning, the term was coined by Arthur Samuel in 1959 [139] who defined it as "the field of study in which computers are given the ability to learn without be explicitly programmed". However, because the term "learn" is very broad, it must be bounded, Mitchell [115] operationalizes it like this: "A computer program learns from an experience E with respect to some type of task T and measure of performance P, if its performance in tasks in T, measured by P, improves with the experience E". Goodfellow et al. [61] list examples of tasks, performance measures and experiences more used in the area of machine learning. Next, a description of these three concepts is given starting in first place with *tasks*.

The most common tasks performed in machine learning are:

- *Classification*. It is perhaps the most important type of task, in classification the computer program must select a category of a set of size k for each of the entries it receives, represented through a vector of n dimensions. To perform this task, the learning algorithm must obtain a model that usually consists of a function $f : \Re^n \to \{1,\ldots,k\}$. So, when $y = f(\mathbf{x})$ the model assigns to the entry represented by $\mathbf{x}$ a category (class) identified with the numeric code y. There are numerous cases where classification algorithms have been successfully applied, for example in the detection of undesired mail (spam), it is possible to use a classification algorithm to determine if an email, represented through a vector, belongs to the category "spam" or instead is a desirable email and belongs to the category "non-spam" [29]. Algorithms that perform this type of task are known as *classifiers*.

- *Variants of the classification*. There are numerous variants to the classical problem of classification described in the previous paragraph, Hernández-González et al. [74] proposes a taxonomy to organize these numerous proposals, of which it is possible to mention some cases. A first variant occurs when the input data is not complete, in this case for the learning algorithm it is not enough to obtain a single function that maps between the inputs and the label, but it needs to produce a set of functions to apply them to the different subsets of your entries with missing data. Another variant to consider is given when the result of the classification is not a single label, but several, this is known as multi-label classification [150]. Two possible cases can be given, in the first the output is represented as a set of labels, in the second the output is a probability distribution along the set of labels.

- *Regression*. In this type of task, the computer program is required to produce a prediction in the form of a real number. In order to give an answer, the learning algorithm must obtain a function $f : \Re^n \to \Re$. A real example of this type of task is given in the prediction of future prices or quantities for an inventory.

- *Transcription and translation*. In this case, the computer program observes unstructured data such as images, audio waves, text in some natural language, etc. and it is expected to

produce a structured output. A classic example of this is the field of *speech recognition* [131], in which the program receives audio waves with spoken language and is expected to return a sequence of words corresponding to the transcription of what was said by the voice. Another similar example is automatic translation, in which a text string is received in a natural language such as Spanish and an equivalent text string is produced in another language such as French.

- *Detection of anomalies*. In this type of task, the computer program carefully examines a set of events or objects and is able to identify when it finds one that is unusual [24]. In practice, this task is commonly applied by the financial entities that administer credit cards, in this case the events are the regular purchases made by the card user and any atypical purchases that are detected are used to block the card and thus prevent possible fraud.

- *Analysis of groups*. In group analysis, the task is to separate a set of objects into different groups so that the objects that are in the same group are more similar (using some measure) to each other than the objects of different groups. Group analysis has demonstrated its ability to reveal hidden structures in biological data, and has particularly helped to investigate and understand the activities of genes and proteins that had not previously been characterized [159].

- *Synthesis*. Synthesis is made when the program is requested to produce new data based on those that already exist. This task is useful in multimedia applications when it is tedious or expensive to generate large volumes of data manually. In the field of video games, it has special utility in the automatic generation of very large objects or landscapes [103].

- *Elimination of noise and missing attributes*. The elimination of noise occurs when the automatic learning algorithm receives as input a corrupt instance $\tilde{\mathbf{x}} \in \Re^n$ by some unknown process, the task is to predict the correct instance $\mathbf{x}$ from the corrupt version $\tilde{\mathbf{x}}$. The elimination of missing attributes occurs, as the name implies, when an instance $\mathbf{x} \in \Re^n$ with missing x_i attributes is received. The task is to predict the values for these attributes.

Second, with respect to *performance measures*, in the case of classification and transcription tasks, the most commonly used measure is *accuracy*. This is simply the proportion of instances for which the model produces the correct result. Analogously, the *error rate* can be obtained as the proportion of instances for which the incorrect result occurs. In addition to these measures, others that are commonly used are obtained from the confusion matrix produced by the model, these are the rates of *true positive and true negative* that correspond to the proportion of instances that were correctly classified as positive and negative respectively, and the rates of *false positives and false negatives*, which refer to the proportions of instances that were incorrectly classified as positive and negative respectively.

From the four rates obtained from the confusion matrix, several metrics are derived, the most commonly used are: *precision, sensitivity, specificity, completeness and F-value* [22, 158]. Although these metrics are initially applied in the *binary classification*, when there are only two labels ($k = 2$), it is possible to generalize them for the case of multiple classes ($k > 2$) using procedures known as micro and macro averages.

Performance is usually measured using a different dataset (test set) than the used for training the model (train set), this is because commonly the main objective of a model is that it is able to generalize to different data coming from the same probability distribution. A very common issue with generalization is known as *overfitting*, it occurs when a model is tightly adjusted to the training data but has poor performance with test data.

Ultimately, according to the *experience* from which the program performs the learning, machine learning can be divided into two broad categories *(i) unsupervised learning and (ii) supervised learning*. In general, the experience that the machine learning algorithms go through is represented by a set of data consisting of a collection of instances with specific attributes according to the task to be performed.

The *unsupervised learning* algorithms attempt to learn useful properties about the structure of the dataset. Usually, is interesting to know, even if implicitly, the probability distribution that generated the dataset. The tasks related to this type of experience are the synthesis, the elimination of noise, the analysis of groups and the elimination of missing attributes. More formally, it is possible to define the unsupervised learning experience as a matrix $\mathbf{M} \in \Re^{n \times m}$, where n is the number of instances or objects that make up the dataset and m is the number of attributes of each instance.

The second major category of machine learning is the *supervised learning*, here the experience consists of a dataset in which each of the instances is associated with a label or class. Classification tasks are carried out with this type of experience. More formally, in supervised learning, the algorithm in addition to having the matrix $\mathbf{M}$, has a vector $\mathbf{y} \in \Re^n$ containing the numeric code of the labels corresponding to each of the n instances of the dataset.

Moreover, the different variants of the classification task also include different types of experiences, for example in the basic scheme of *semi-supervised learning* [26] only part of the dataset has labels, although the rest is also used for learning. Another case of semi-supervised learning is that of *multi-instance learning* where the labels are assigned to groups instead of the individual instances [168].

2.3 Data Preprocessing

Considering again the stages of the KDD process described in Section 2.1, although the data mining stage is the central stage, additional steps such as the understanding and specification of the problem, data preprocessing and evaluation are essential to be able to ensure the obtaining

of valuable knowledge from the data. The "blind" application of data mining can be a dangerous activity and can easily lead to the discovery of meaningless and invalid patterns [51].

With regard to the preprocessing stage, this may involve a considerable number of sub-steps of various kinds, García et al. [57] group these sub-steps into two categories *(i) data preparation* and *(ii) data reduction*. Next, each one of them are described.

- *Data preparation*. This category groups those sub-steps that allow converting data that in its actual state is not possible to use directly in the subsequent stage of data mining. The sub-steps grouped here are:
  - *Data cleaning*. It is usually done with human intervention since it requires the understanding of the domain of the problem, eliminating data that may be unnecessary or incorrect. In addition, tasks such as the detection and elimination of noise and missing attributes are performed, in some cases with the help of machine learning algorithms.
  - *Transformation of data*. This sub-step, similarly to the previous one, requires considerable human intervention, here the data is converted or consolidated so that the mining process is more efficient, some of the tasks that can be performed are: the construction, aggregation or summary of features and the smoothing and normalization of the data.
- *Data reduction*. This category includes a group of techniques that in some way reduce the amount of original data that the data mining algorithm must process. It differs from the previous category in that here the input data is already in a valid state in order to serve as input to a data mining algorithm without obtaining errors related to the values provided. For this reason, it could be considered an optional stage. However, considering the accelerated growth of the datasets that are currently experienced and the constraints according to the algorithmic complexity of most data mining methods (see Section 3.1), in many cases the reduction of data becomes a requirement for the execution of these algorithms. The following sub-steps are placed here:
  - *Feature selection*. This achieves the reduction of the dataset through the elimination of redundant or irrelevant attributes, generally through algorithms that require less human intervention during the preparation of data. This sub-step of the KDD process constitutes a essential topic of this thesis work, which is why it is described in more detail in the following section.
  - *Instance selection*. As the name implies, the reduction of the dataset is done by selecting the best instances of all the available ones. This can be done in order to improve the execution speed of the algorithm and the memory requirements or for more complex cases such as reducing the overfitting of the model or treating the imbalance in the dataset [39].

– *Discretization*. Discretization is the process used to convert data from a continuous domain to a discrete domain. To do this, the continuous values are separated into a finite number of ranges and each range is assigned a discrete value. This task can actually be classified as part of the preparation of the data, since there are numerous data mining algorithms that do not support continuous data and therefore discretization becomes a requirement. However, the discretization process also entails a reduction in the spectrum of values of the dataset, which is why it is included in this category [56].

2.4 Feature Selection

As said before, feature selection is a essential topic of this thesis work, as evidenced by its title. The previous sections have been included in order to adequately contextualize its position within the KDD process and its relationship with the data mining and machine learning fields. According to Guyon and Elisseeff [66] the objective of feature selection is threefold: (i) to improve the performance of predictive models, (ii) to make them faster and more effective with respect to their cost in resources and (iii) to allow a better understanding of the underlying process that generated the data.

In addition, feature selection allows to alleviate the negative effects caused by the *curse of the dimensionality*, a term introduced by Bellman [11] and which refers to the fact that a normal increase of the dimensions (features) in the dataset leads to a exponential increase of the search space and the growth in the probability of obtaining invalid models. Some data mining techniques are more prone to suffer from the curse of dimensionality, for example decision trees and instance-based learning. Finally, another positive effect of feature selection is to reduce the cost of data acquisition, which is evident when it allows to avoid collecting features of an instance that have been determined to be irrelevant.

Formally, if X is the set of features, feature selection consists in choosing (following some defined criterion) a subset $S \in \mathscr{P}(X)$, where $\mathscr{P}(X)$ is the power set of X.

2.4.1 Categorization

Similar to machine learning algorithms, it is possible to perform an initial categorization of feature selection methods in supervised and unsupervised methods, according to the presence or absence of labels for the instances in the dataset. Unsupervised methods are considered the most complex ones [138]. Mitra et al. [116] classify the unsupervised methods in two categories, the first one refers to the methods oriented to maximize the performance of the analysis of groups and the second refers to the methods that evaluate the attributes according to dependence and relevance measures, under the principle that any extra feature that does not provide enough information beyond what is already represented by the current set of attributes is redundant

and must be eliminated. However, in the current work emphasis is on the supervised methods in which the dataset is labeled. So, from now on, references to feature selection will indeed be references to *supervised feature selection* unless otherwise specified.

Traditionally, feature selection methods have been classified into three categories [66] according to their relationship with the classification algorithms. Figure 2.2 shows the structure of this classification described below:

- *Filters*. Filters methods use metrics to evaluate attributes that do not require the training of a classifier and depend exclusively on the intrinsic properties of the data. In other words, the search in features space is done previously to the classification process. For this reason, they are usually the algorithms that require less processing and memory resources than the rest. In addition, filters are commonly classified in univariate and multivariate, depending on whether the evaluation of the attributes is done individually or collectively, respectively. The multivariate evaluation allows to consider the dependencies and interactions between the attributes but usually has a higher computational cost.

- *Wrappers*. These methods are named this way because they define a search method that "wraps" a classifier and uses it to evaluate the attributes. That is, the search in the features space involves multiple searches in the hypothesis space (made by the classifier). These are typically the most computationally expensive methods because they require the classifier to be trained multiple times in each step of the search, but at the same time, they are generally the methods that lead to better accuracy rates, running the risk of overfitting in some cases [102].

- *Embedded*. These are methods in which the selection of attributes is part of a classifier, they are implemented through the use of objective functions that in addition to considering the quality of the fit of a model, also penalize that it is made up of many variables. They are proposed with the objective of avoiding the computational efficiency problem of the wrapper methods since they do not require the training of multiple classifiers. In this case, the search in the features space is performed together with the search of the hypothesis.

In addition to the previous categorization, it is possible to classify feature selection according to the following four elements: (i) the output they produce, (ii) the search direction, (iii) the search strategy they follow, and (iv) the metrics they use to evaluate the attributes. According to the output they produce, it is possible to define two subcategories:

- *Feature Ranking*. Methods in this subcategory produce an output that consists of an ordered list of features according to their importance depending on the metric used. In order to proceed with feature selection the first u features of the list are chosen. However, the problem with these is that in many cases there is no defined number of features to choose from and there are no direct procedures for selecting a threshold value [57]. Many of these

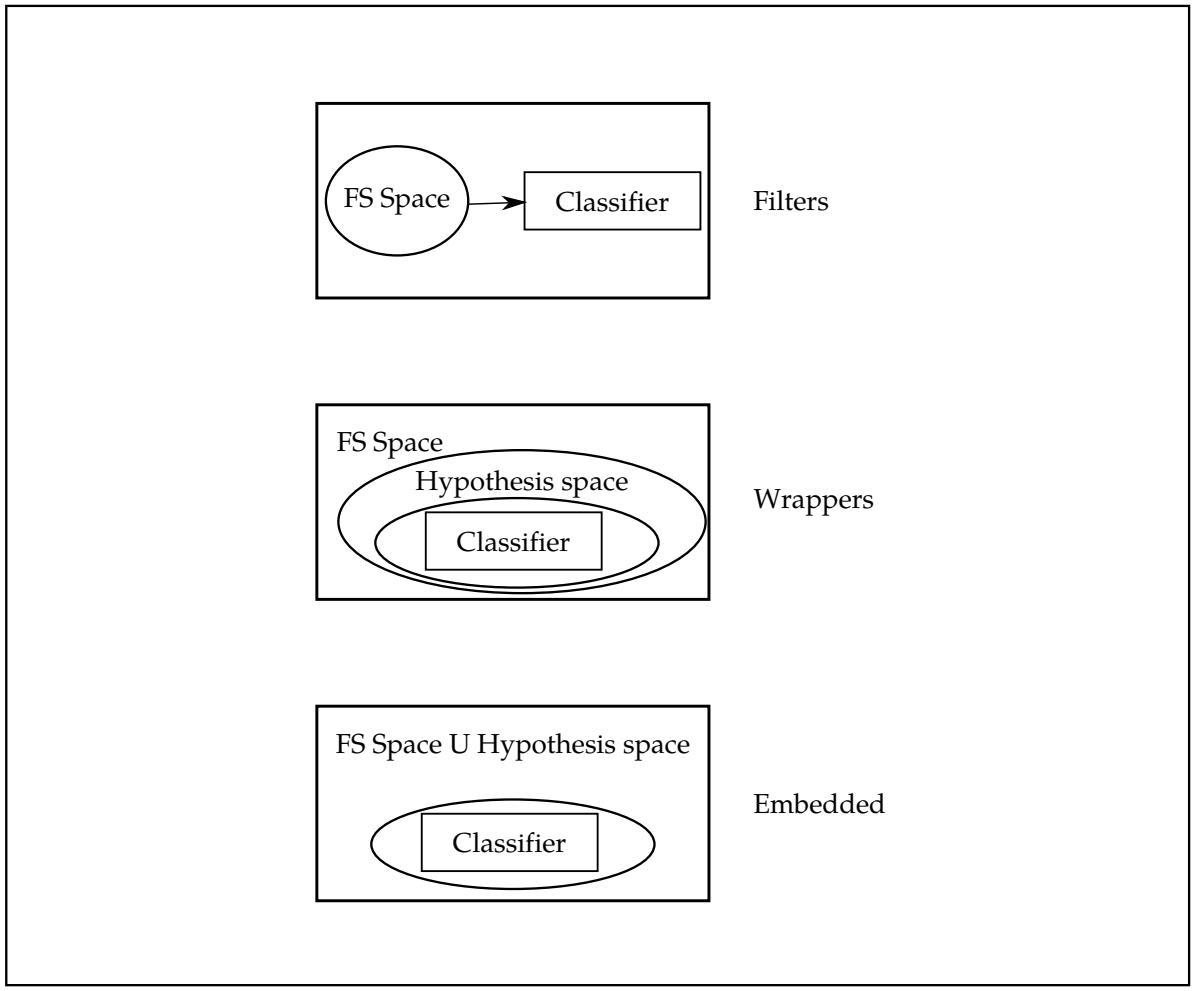

Figure 2.2: Feature selection methods main classification [138]

methods assign a weight value to each feature and then use these weights to produce the ranking.

- *Subset Selection*. The output of this type of method consists of a subset of the original features. Within this, no distinction of importance is usually made, simply the features that are considered most important are placed within the subset and the rest is left out. These methods have the advantage of not requiring a previous definition of the number of features to be selected nor a threshold to make the selection.

In reference to the search direction that is followed, according to Liu and Yu [99] it is possible to mention four categories, but not before clarifying that not all algorithms of selection of features need to perform a search, some algorithms, for example univariate filters, do not need more than going through the set of features and applying the corresponding metric to each of them according to their values.

- *Forward Search*. This type of search begins with an empty set of features that is increased by selecting the next best feature according to some criteria. The search may end either because the number of selected features has already reached a threshold value or because all the possible subsets have already been traversed.

- *Backward Search*. Conversely to the previous one, it starts with the complete set of features that are eliminated one by one according to some criterion that indicates which is the least important so that in the end the last feature to be eliminated is considered the most relevant of all. In addition, finalization criteria such as the number of deleted features are commonly used.

- *Bidirectional Search*. The bidirectional generation consists simply in the parallel execution of the two previous searches in order to complete the search faster. Then the results have to be merged in some fashion.

- *Random Search*. In order to avoid stagnation in a local optimum, the search starts with a random set and the decision to add or remove features is also made randomly.

Given a search direction, this should be combined with a search strategy, García et al. [57] classify them and describe three categories:

- *Exhaustive Search*. This search involves the exploration of all possible solution subsets, that is, if X is the initial set, it involves traversing all members of $\mathscr{P}(X)$, and if $|X| = n$, then $|\mathscr{P}(X)| = 2^n$, so this search grows exponentially with the number of features n, becoming unfeasible in most cases. However, it is the only search that guarantees to find the optimal result.

- *Heuristic Search*. Given the unfeasibility of the exhaustive search, this search avoids evaluating all alternatives in $\mathscr{P}(X)$ by creating a set in $\mathscr{O}(n)$ steps, using a heuristic to select the members of the result.

- *Non-deterministic Search*. Also known as random search, it does not follow a certain order but generates random results which are evaluated hoping that each new result is better than the current one. The search usually stops after a time interval has elapsed or when a defined quality level is obtained.

2.4.2 Feature Evaluation Metrics

As mentioned above, filters use different feature evaluation metrics that depend exclusively on the intrinsic characteristics of the data, these metrics can be classified into four categories:

- *Information*. These metrics are based on Shannon's Information Theory. They use the concept of uncertainty and evaluate the features according to their capacity to reduce

uncertainty with respect to the class. A very important concept that conforms the basis of information theory is that of *entropy* of a discrete random variable by itself or given another discrete random variable, both depicted in Equations 2.1 respectively. The entropy is a measure of the amount of uncertainty a random variable holds, for example if X is a Bernoully random variable with $p = 0.9$, it will a have an entropy of $H(X) \approx 0.47$ but if the amount of uncertainty is incremented by setting $p = 0.5$ then the entropy will raise to $H(X) = 1.0$.

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$
$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log_2 p(x|y) \tag{2.1}$$

- *Distance*. Comparing with the previous metrics, these ones instead of select features that reduce the uncertainty, prefer features that increase the distance between the classes, for this reason they are also known as separability metrics. One of the most used distance measurements in the Euclidean Distance DE defined between two points $\mathbf{X}, \mathbf{Y}$ in $\Re^n$ as $DE = \left[\sum_{i=1}^{n} (x_i - y_i)^2\right]^{1/2}$.

- *Correlation*. Correlation metrics evaluate the level of association between two variables. These associations are measured between different features as well as between the features and the class. Two features that are closely associated are often considered redundant so that one of the two can be eliminated. On the other hand, features that have high correlation with the class are preferred because of their predictive potential. One of the most commonly used correlation measures is the Pearson coefficient defined as $\rho_{X,Y} = \frac{cov(\mathbf{X},\mathbf{Y})}{\sigma_X \sigma_Y}$, where cov and σ represent the functions of covariance and variance respectively. However, one of the disadvantages of the Pearson coefficient is that it only allows to detect linear correlations between the variables, therefore other measures such as the Symmetric Uncertainty [128] are used.

- *Consistency*. These metrics are applied by reducing the number of features and at the same time minimizing the number of inconsistencies in the data, according to [33] an inconsistency is found in dataset when two instances have the same values in their features but belong to different classes.

In addition to the metrics used by filters, wrappers use different measures to evaluate the performance of the classifiers they train, however since this thesis work is focused on the design of filters, wrappers metrics are not mentioned here.

2.4.3 Evaluating Feature Selection

From the numerous categories and evaluation metrics mentioned in the previous sections it is easy to infer that there are numerous methods for feature selection and, as expected, they do not behave in the same way, since some are more convenient than others depending on the characteristics of the data. For example, Bolón-Canedo et al. [15] perform a comparison of different method of attribute selection considering their capabilities to handle the following situations:

- A lot of correlation and redundancy between the features.
- Non-linear dependencies between features.
- Noise in the features and in the class.
- Very low proportion of instances with respect to number of features.

This comparison is made using synthetic data, so it is possible to evaluate the process according to an expected result. However, in practice, the ideal set of features is not really known and the different methods applied may return different results. In order to select one of the necessary results, the evaluation will be directed by the objective for which the selection of features is being carried out, which, as mentioned at the beginning, is triple. Below, three evaluation criteria according to this objective are described.

1. *Predictive power*. By reducing the number of features it is possible to achieve that some data mining algorithms improve their predictive performance, so that the process of selection of features under this criterion is evaluated by measuring the performance of the subsequent predictive algorithm.
2. *Interpretability*. There are models in data mining that, in addition to their predictive capabilities, also provide a summary representation of the data that can be interpreted by an expert in the area. By reducing the number of features it is possible to obtain representations that are simpler and easier to understand. The evaluation of this criterion depends on a measure of complexity that must be adapted to the type of model generated. For example, in the case of a decision tree, its complexity could be measured according to its number of branches, leaves and nodes.
3. *Reduction of costs*. Obtaining a reduced set of features also has as a common consequence a reduction in the consumption of computational resources such as processor time and memory. Therefore, when evaluating this criterion, the reduction in the consumption of these resources of the subsequent data mining algorithms is measured. However, this objective is not usually pursued independently without being linked to at least one of the previous two. For example, reduce the cost and maintain predictive power.

Apart from these three basic criteria, it is also possible to mention other practical factors that may be important to consider when selecting a feature selection algorithm:

- *Algorithmic complexity*. Due to the marked growth trend in the size of the datasets, in some cases it is possible that the feature selection algorithm may not be able to process all of them in a reasonable time or that the memory requirements are greater than those available so the algorithm cannot be executed. For these reasons, in these cases, filters are preferred over wrappers.

- *Stability*. Kalousis et al. [83] define the stability as the robustness of the results produced with respect to the differences in training sets taken from the same probability distribution. The lack of stability in a method can become an undesirable factor in fields such as biology where it is desired that the set of selected features do not have radical changes with small changes in the training set since it is common that considerable research effort will be made on these features.

2.4.4 Filter-based Feature Selection Algorithms

In this section, some of the most prominent filter-based feature selection algorithms are presented as examples of this important category of algorithms. The algorithms were selected from recent surveys in the field [15, 25, 96, 140]. A special emphasis is placed in the discussion of the two final algorithms, since as mentioned in Chapter 1 they constitute the main conceptual basis of the contributions of this dissertation.

2.4.4.1 Fisher Score [45]

The Fisher Score uses a criteria based on distance, the features selected are those whose values on instances of the same class are small and are large on instances of different classes. Fisher Score is a univariate filter, it produces a feature ranking by scoring each individual feature using the Fisher criterion shown in Equation 2.2:

$$F(f_i) = \frac{\sum_{j=1}^{c} n_j(\mu_{i,j} - \mu_i)^2}{\sum_{j=1}^{c} n_j \sigma(i,j)^2} \tag{2.2}$$

where n_j, μ_i, $\mu_{i,j}$ and $\sigma(i,j)^2$ indicate the number of samples in class j, mean value of feature f_i, mean value of feature f_i for samples in class j and variance value of feature f_i for samples in class j, respectively. Given the univariate nature of the Fisher score, it is incapable of removing redundant features, to overcome this issue Gu et al. [65] propose the Generalized Feature Score, a multivariate technique that produces a features subset that maximizes the lower bound of the original Fisher score.

2.4.4.2 Information Gain [95]

As the name implies, Information Gain uses a information theory based measure to evaluate features known as Mutual Information, shown in Equation 2.3. This metric is used to measure the amount of dependence between two random variables X (a determined feature) and Y (the class) by using their entropy and conditional entropy. Similar to the previous, it is a univariate filter that produces a feature ranking as output.

$$I(X,Y) = H(X) - H(X|Y) \tag{2.3}$$

2.4.4.3 Minimum Redundancy Maximum Relevancy [125]

Minimum Redundancy Maximum Relevancy known as mRMR is a multivariate filter that produces a feature ranking, this ranking is based on relevance of the features with respect to the class penalizing at the same time the redundancy of features. The relevance of a feature is based on the mutual information it shares with the class and its redundancy is obtained with the mutual information it shares with the rest of selected features.

2.4.4.4 Fast Correlation Based Filter [160]

Different to the previous ones, the Fast Correlation-Based Filter (FCBF) does not produce a feature ranking but a features subset. It is also a multivariate filter since it considers relations between features trying to reduce redundancy. FCBF uses an information theory based correlation measure known as Symmetrical Uncertainty [128] depicted on Equation 2.4, this measure is capable of detecting linear a non-linear correlations between two discrete random variables being them features or the class. A valuable characteristic of the symmetric uncertainty is its symmetry, that is $SU(X,Y) = SU(Y,X)$ this is useful when calculating associations between features since none of them can be identified as the class.

$$SU(X,Y) = 2 \cdot \left[\frac{H(X) - H(X|Y)}{H(Y) + H(X)} \right] \tag{2.4}$$

2.4.4.5 Consistency-based Filter [33]

As suggested by its name, the Consistency-based Filter uses a consistency measure known as the *inconsistency rate* that is calculated based in the concept of *pattern*, a pattern is simply the set of features values a specific instance has, that is, an instance without the class value. The inconsistency rate of a feature subset is determined first by calculating the *inconsistency count* for each pattern on the subset. This count is equal to the number of times the pattern appears in the dataset minus the largest number of times it appears among different class labels. For example, if a feature subset S has a pattern p that appears in n_p instances out of which $c1$ instances have

class $label_1$, $c2$ have $label_2$, and $c3$ have $label_3$ where $c1+c2+c3=n_p$, then if $c3$ is the largest among the three, the inconsistency count is $n-c3$. Once the inconsistency count is known, the inconsistency rate of the feature subset is simply the sum of all inconsistency counts over all patterns of the subset divided by the number of instances the dataset has.

2.4.4.6 ReliefF [88]

ReliefF is multivariate filter that produces a feature ranking, it is an extension of the original Relief algorithm [86]. Both algorithms share the central idea that consists in evaluating the quality of the features by their ability to distinguish instances from one class to another in a local neighborhood, i.e., the best features are those that contribute more to increase distance between different class instances while contribute less to increase distance between same class instances. The original Relief algorithm was designed for binary class problems, and ReliefF extends its capabilities for working with multi-class, noisy and incomplete datasets. ReliefF has been recognized for is good tolerance to noise, both in labels and inputs and for detecting non-linear interactions between features and the class [15].

Algorithm 1 ReliefF [88, 135]

calculate prior probabilities $P(C)$ for all classes
set all weights $W[A] := 0.0$
for $i = 1$ **to** m **do**
  randomly select an instance R_i
  find k nearest hits H_j
  for all classes $C \neq cl(R_i)$ **do**
    from class C find k nearest misses $M_j(C)$
  end for
  for $A := 1$ **to** a **do**
    $\overline{H} := -\sum_{j=1}^{k} diff(A, R_i, H_j)/k$
    $\overline{M} := \sum_{C \neq cl(R_i)} \left[\left(\frac{P(C)}{1-P(cl(R_i))} \right) \sum_{j=1}^{k} diff(A, R_i, M_j(C)) \right] /k$
    $W[A] := W[A] + (\overline{H} + \overline{M})/m$
  end for
end for
return W

Algorithm 1 displays ReliefF's pseudo-code, mostly preserving the original notation used in [135]. As it can be observed, it consists of a main loop that iterates m times, where m corresponds to the number of samples from data to perform the quality estimation. Each selected sample R_i equally contributes to the a-size weights vector W, where a is the number of features in the dataset. The contribution for the A-th feature is calculated by first finding k nearest neighbors of the actual instance for each class in the dataset. The k neighbors that belong to the same class as the actual instance are called *hits* (H), and the other $k \cdot (c-1)$ neighbors are called *misses* (M), where c is the total number of classes, and $cl(R_i)$, represents the class of the i-th sample.

Once the neighbors are found, their respective contributions to A-th feature are calculated. The contribution of the hits collection $\overline{H}$ is equal to the negative of the average of the differences between the actual instance and each hit. It should be noted that this is a negative contribution because only non desirable features should contribute to create differences between neighbor instances of the same class. Analogously, the contribution of the misses collection $\overline{M}$ is equal to the weighted average of the differences between the actual instance and each miss. This is a positive contribution because good features should help to differentiate between instances of a different class. The weights for this summation are defined according to the prior probability of each class, calculated from the dataset. Finally, it is worth mentioning that adding $\overline{H}$ and $\overline{M}$ and then dividing both by m simply indicates another average between the contributions of all m samples. Since the *diff* function returns values between 0 and 1, the ReliefF's weights will be in the range $[-1,1]$, and must be interpreted in the positive direction: the higher the weight, the higher the corresponding feature's relevance.

The *diff* function is used in two cases in the ReliefF algorithm. The obvious one is between lines 10 and 11 to calculate the weight. It is also used to find distances between instances, defined as the sum of the differences over every feature (Manhattan distance). The original *diff* function used to calculate the difference between two instances I_1 and I_2 for a specific feature A is defined in (2.5) for nominal features, and as in (2.6) for numeric features. However, the latter has been proved to cause an underestimation of numeric features with respect to nominal ones in datasets with both types of features. Thereby, a so-called ramp function, depicted in (2.7), was proposed to deal with this problem [77]. The idea behind it is to relax the equality comparison on (2.6) by using two thresholds: t_{eq} is the maximum distance between two features to still consider them equal, and analogously, t_{diff} is the minimum distance between two features to still consider them different. Their default values are set to 5% and 10% of the feature's value interval respectively. In addition, there are other versions of the *diff* function to deal with missing data. However, since the datasets chosen for the experiments in this work do not have missing values, they are not considered here.

$$diff(A,I_1,I_2) = \begin{cases} 0 & \text{if } value(A,I_1) = value(A,I_2), \\ 1 & \text{otherwise} \end{cases} \tag{2.5}$$

$$diff(A,I_1,I_2) = \frac{|value(A,I_1) - value(A,I_2)|}{max(A) - min(A)} \tag{2.6}$$

$$diff(A,I_1,I_2) = \begin{cases} 0 & \text{if } d \le t_{eq}, \\ 1 & \text{if } d > t_{diff}, \\ \frac{d - t_{eq}}{t_{diff} - t_{eq}} & \text{if } t_eq < d \le t_{diff} \end{cases} \tag{2.7}$$

2.4.4.7 Correlation-based Feature Selection [71]

Correlation-based Feature Selection (CFS) is categorized as a subset selector, it evaluates subsets rather than individual features. For this reason, the CFS needs to perform a search over candidate subsets, but since performing a full search over all possible subsets is prohibitive (due to the exponential complexity of the problem), a heuristic has to be used to guide a partial search. This heuristic is the main concept behind the CFS algorithm, and, as a filter method, the CFS is not a classification-derived measure, but rather applies a principle derived from Ghiselly's test theory [59], i.e., *good feature subsets contain features highly correlated with the class, yet uncorrelated with each other*.

This principle is formalized in Equation (2.8) where M_s represents the merit assigned by the heuristic to a subset s that contains k features, $\overline{r_{cf}}$ represents the average of the correlations between each feature in s and the class attribute, and $\overline{r_{ff}}$ is the average correlation between each of the $\binom{k}{2}$ possible feature pairs in s. The numerator can be interpreted as an indicator of how predictive the feature set is and the denominator can be interpreted as an indicator of how redundant features in s are.

$$M_s = \frac{k \cdot \overline{r_{cf}}}{\sqrt{k + k(k-1) \cdot \overline{r_{ff}}}} \tag{2.8}$$

Equation (2.8) also posits the second important concept underlying the CFS, which is the computation of correlations to obtain the required averages. In classification problems, the CFS uses the symmetrical uncertainty measure [128] previously shown in Equation (2.4). This calculation adds a requirement for the dataset before processing, which is that all non-discrete features must be discretized. By default, this process is performed using the discretization algorithm proposed by Fayyad and Irani [52].

The third core CFS concept is its search strategy. By default, the CFS algorithm uses a best-first search to explore the search space. The algorithm starts with an empty set of features and at each step of the search all possible single feature expansions are generated. The new subsets are evaluated using Equation (2.8) and are then added to a priority queue according to merit. In the subsequent iteration, the best subset from the queue is selected for expansion in the same way as was done for the first empty subset. If expanding the best subset fails to produce an improvement in the overall merit, this counts as a *fail* and the next best subset from the queue is selected. By default, the CFS uses five consecutive fails as a stopping criterion and as a limit on queue length.

The final CFS element is an optional post-processing step. As stated before, the CFS tends to select feature subsets with low redundancy and high correlation with the class. However, in some cases, extra features that are *locally predictive* in a small area of the instance space may exist that can be leveraged by certain classifiers [70]. To include these features in the subset after the search, the CFS can optionally use a heuristic that enables inclusion of all features whose

correlation with the class is higher than the correlation between the features themselves and with features already selected. Algorithm 2 summarizes the main aspects of the CFS.

Algorithm 2 CFS [71]

```
1: Corrs := correlations between all features with the class
2: BestSubset := ∅
3: Queue.setCapacity(5)
4: Queue.add(BestSubset)
5: NFails := 0
6: while NFails < 5 do
7:    HeadState := Queue.dequeue {Remove from queue}
8:    NewSubsets := evaluate(expand(HeadState),Corrs)
9:    Queue.add(NewSubsets)
10:   if Queue.isEmpty then
11:      return BestSubset {When the best subset is the full subset}
12:   end if
13:   LocalBest := Queue.head {Check new best without removing}
14:   if LocalBest.merit > BestSubset.merit then
15:      BestSubset := LocalBest {Found a new best}
16:      NFails := 0 {Fails must happen consecutively}
17:   else
18:      NFails := NFails + 1
19:   end if
20: end while
21: {Optionally add locally predictive features to BestSubset}
22: return BestSubset
```

CHAPTER 3

BIG DATA AND OTHER RELATED TERMS

The chapter ahead represents the second part of the background concepts. It starts with a discussion about the term *"Big Data"*, its meaning and implications. Next, its lists and discusses other important terms that have grown in popularity with big data or that form part of the history of it, such as *"Data Science"* and *"Business Intelligence"*. The aim of this chapter is to define and clarify the relations between all of these terms and the data mining and machine learning concepts discussed in the previous chapter.

3.1 Big Data

According to Diebold [41], the term big data probably appeared during the mid 90's lunch conversations in Silicon Graphics where John Mashey worked as a researcher. A proof of this are the slides of the presentation titled "Big Data... and the next wave of Infrastress" [109] where Mashey discussed the importance of being aware about the increasing demand of information services and the stress over hardware infrastructure: storage, processing, memory and network that this demand was going to cause.

However, the term has only recently become popular, according to Gandomi and Haider [53] its popularization started in 2011, supported by IBM and other leading technology companies. The term suddenly appeared and was quickly accepted by many sectors, but the academic domain was somewhat left behind in such a manner that the term became widespread without even having a commonly accepted definition [2]. Moreover, between the amount of discussion the term has generated, is possible to state some properties about it:

Big Data is a trend. In the current information era, were the information has a tremendous economical value, it is now clear for everyone that organizations and individual researchers

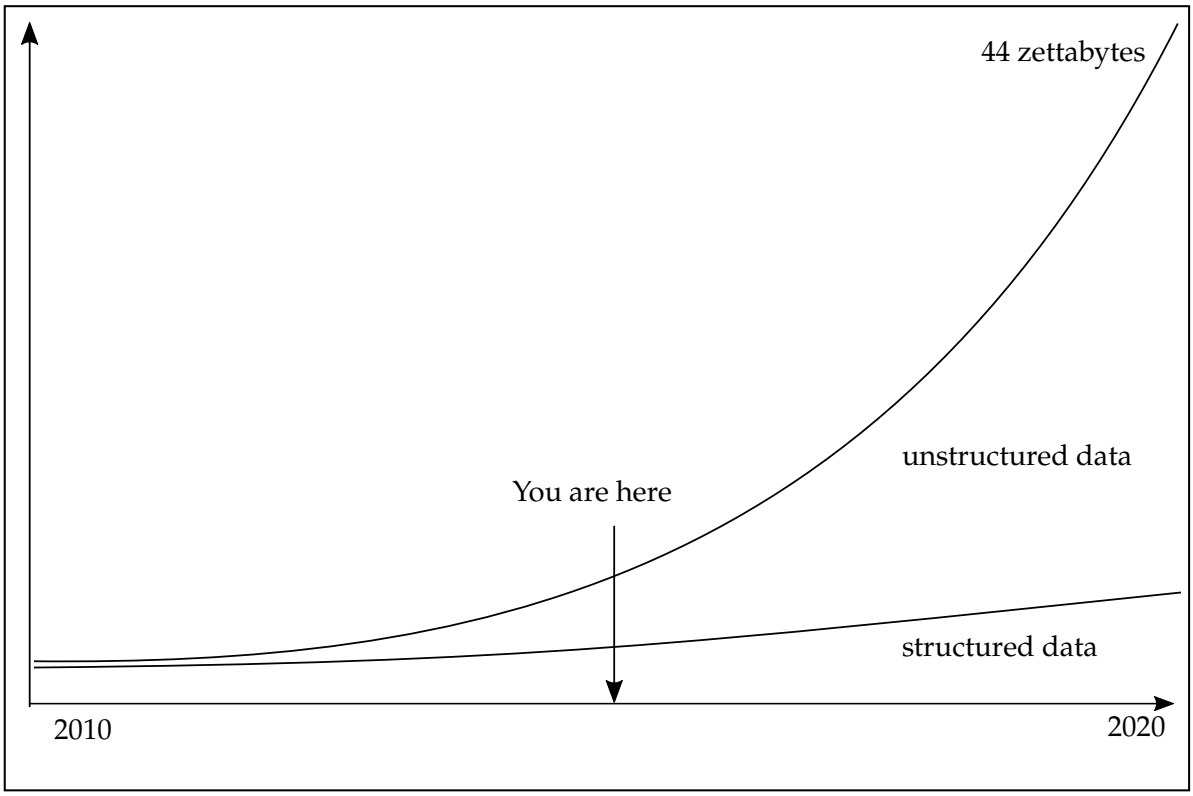

Figure 3.1: Exponential growth of the data universe [67]

around the world will try to collect and analyze all the data about their own activities and the environment that surrounds them in order to produce valuable information. Figure 3.1 depicts the exponential growth of the data universe that is been experimented, the total amount of data the humanity is storing is going from 4.4 Zettabytes in 2015 to 44 Zettabytes in 2020, doubling in size every two years according to the IDC / EMC [78] report in 2014.

Big Data is a property of the data. Cox and Ellsworth [31] wrote the first article in the ACM library that uses the term big data. They define the *"problem of big data"* as occurring when the datasets are too large for being stored in local memory, storage or even remote storage. Currently, this problem concerns not only the storage but also emerges when the datasets are too large for being processed, for example by data analysis tools. This coincides with the fact that most of the current stored data is non-structured data (videos, audio, images, etc.) [54] and this type of data has always been a difficult task for analysis tools.

Big Data is an opportunity. The big data phenomena driven by the continuous improvements in all information systems technologies such as processors, memories, disk drives and networking, and the widespread of these technologies in the form of devices such as smart-phones, laptops, desktops, servers, network devices and a new gamma of Internet-connected devices labeled as the Internet Of Things (IoT), has been clearly identified as a great opportunity to generate more value

from the data than before [107] and to help advance in practically all sectors from government [85] to business management [110]. New applications are continuously been published in all fields of science, such as biology [108], health care [149], social sciences [121] and civil [5] and electric engineering [80], to give some examples. Regarding to data mining and machine learning, it is widely known that the quality of many models and their predictive performance can be improved by increasing the amount of data used for training [82], in such a way that big data has become crucial in the recent advances in this area.

In 2001, Doug Laney, a recognized analyst at the prestigious information technologies consulting firm Gartner, published a technical report [91] describing three dimensions in which data management challenges had been expanded: Volume, Velocity and Variety . These three dimensions have been called the three V's of big data and have become a common framework to describe it. They are reviewed next:

1. Volume. Refers to the fact already discussed below, there is more data than ever before, and its size continues to increase making it a challenge for the actual infrastructure to store it and process it to generate value.

2. Velocity. This indicates that the data is arriving in a continuous stream and there is interest in obtaining useful information from it in real time. Cisco Systems in an article titled "The Zettabyte Era" [27] mentions that 2016 was the first year when more than 1 Zettabyte was transmitted over the Internet and forecasts that this amount will triplicate over the next five years.

3. Variety. This dimension alludes to the diverse amount of sources where the data is obtained and the many formats it can have. As an example of this, according to van Rijmenam [154] the famous retail corporation Walmart uses about 200 sources including data from weather, product sales status, pricing, inventory and many more with the aim of forecasting the needs of their 250 million weekly customers.

When contrasting these three dimensions with the three properties about big data that were mentioned before, is possible to say that the three dimensions can be included in the first two mentioned properties: big data is a trend and a property of the data. Moreover, the three previous dimensions form only the base of the current Gartner definition of big data: "high volume, velocity and variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making". For this reason, is it possible to add at least a fourth V, referring to value, indicating that the enhanced insight and decision making leads to generate scientific or business value. And with this, the third mentioned property can be fulfilled: big data is an opportunity to generate value.

3.2 Big Data Related Terms

In a similar way to the term big data, other related terms have previously appeared suddenly, and have proposed similar difficulties to the scientific community in order to properly define them [73], mainly because their quick adoption also brings to different conceptions. Four of such terms are: business intelligence, analytics and data science. This section, attempts to describe them by listing some of their common definitions and establishing their relations to each other and with big data.

3.2.1 Business Intelligence

According to Davenport [35], the term business intelligence (BI) became popular in the late 1980s, encompassing a wide array of software and processes designed for collecting, analyzing and disseminating data with the final aim of better decision making. Similarly, the Gartner dictionary defines it as an umbrella term that includes the applications, infrastructure and tools, and best practices that enable access to and analysis of information to improve and optimize decisions and performance. Its easy to observe that the term business intelligence already reflects the desire of obtaining value from data using different methods and software tools that was previously mentioned as the opportunity of big data. However, in practice the new tools that are being used with the same final objective of obtaining this value are progressively being called "big data tools" instead of "BI tools" willing to differentiate from the other in the properties of the data they handle (volume, velocity and variety), and in the analysis methods used, methods that are changing from being only descriptive to become more predictive by using machine learning models.

3.2.2 Analytics

Rose [136] mentions that the term analytics as used today was arguably introduced in Davenport [35]. But more than that, Rose [136] discusses how the term emerged surrounded by uncertainty as it did not had a clear definition nor clearly defined relations with disciplines as statistics, computer science and operations research. Later, the author identifies three different definitions for the term:

1. *Analytics as a synonym for statistics*. When used as in website analytics, it only refers to statistics of usage as how many clicks or views the website has.

2. *Analytics as a synonym for data science*. The discussion about this definition is deferred until the term data science is addressed below.

3. *Analytics as a quantitative approach to organizational decision-making*. This is the use that Davenport [35] gives to the term. This is the most broadly used definition and under

> this, analytics is just basically another umbrella term to make reference to all quantitative decision disciplines such as: computer science, statistics and operations research, all used in organizations of all types with a wide variety of applications such as: reducing inventories and stock-outs, identifying the best customers (most profitable), selecting prices, employees and improving or developing new products.

Again, it is easy to observe from this last definition of analytics, that the term is intrinsically related to the two previous terms: business intelligence and big data, Davenport [36] makes this relations clear when describing the evolution of analytics, classifying it on three eras:

Analytics 1.0. First of all, is important to mention that using data for making decisions is not a revolutionary idea, is it indeed the natural way of making decisions for humans and other species. However, this first era is marked as beginning during the mid-1950s with the advent of tools that could capture and produce larger quantities of data and discern patterns much faster than the unassisted human mind ever could.

This era is known as the "business intelligence era", were enterprises started using and building software for capturing data in data warehouses and then using it for making reports. However, these reports were centered on describing what happened in the past, offering no explanations or predictions of the future.

Analytics 2.0. This is called by Davenport as the big data era. It started in the mid-2000s when the internet-based and social networks firms such as Google, eBay, LinkedIn, and so on, began to collect and analyze new kinds of information.

This is the era were big data emerged as a trend between online businesses and as a property of the data. This led to the creation of innovative technologies such as Apache Hadoop and Apache Spark for faster data processing and NoSQL databases to deal with the scalability issues and the rise of the storage and analysis of non-structured data.

The skills needed for this era were different from before and a new generation of quantitative analysts was needed, they were called: "data scientists" reflecting the fact that they main job was to study the data (commonly by leveraging the emerging big data tools or by developing their own tools), make new discoveries, communicate them (maybe using visualizations tools) and suggest implications in business decisions. Thereby, a data scientist was identified as combination of a programmer, a statistician, a storyteller and a business consultant.

Analytics 3.0. This era is mainly defined by Davenport as the moment when other large organizations different from the original information centric businesses like Google and Amazon, start to follow suit and subsequently every firm and every industry is able to leverage the increasing amounts of data generated by themselves and available from others to create and reshape products and services from the analysis of this data.

Moreover, analytics have been categorized under three types: descriptive analytics, which makes reports based on past data, predictive analytics, which uses models based past data to predict the future and finally, prescriptive analytics, which uses models to describe optimal

behaviors and the best actions to perform. As can be inferred, the analytics 1.0 era was descriptive in essence, the 2.0 era was descriptive and predictive and in the 3.0 era the emphasis is placed in prescriptive analytics without leaving behind the other two.

3.2.3 Data Science

The third term in the list was data science, this term in a similar way to big data, has gained popularity in the last decade, but its roots can be tracked many decades ago when John Tukey published a visionary paper titled "The Future of Data Analysis" [151] where he introduced the term *"data analysis"* as a superset of statistics and called it a new science. According to Tukey, this new science was driven by four influences:

1. The formal theories of statistics.
2. Accelerating developments in computer and display devices.
3. The challenge, in many fields, of more and ever larger bodies of data.
4. The emphasis on quantification in an increasingly wider variety of disciplines.

Donoho [43] considers that this list is surprisingly modern and that it encompasses all the factors cited in the recent data science initiatives. He also mentions that the idea of having a new field different of statistics has had many detractors that argue that data science is only a rebranding of the centuries old field of statistics.

However, is a fact that the term has become widespread in the recent years mainly after an article published by T. H. Davenport and D.J. Patil in 2012 titled "Data Scientist: The Sexiest Job of the 21st Century" [123], where they present the *"data scientist"* as a new type of professional with the training and curiosity to make discoveries in the world of big data and as an "hybrid of a data hacker, analyst, communicator and trusted advisor". This coincided with insights given by firms as the McKinsey Global Institute [107] predicting that by 2018 the US alone would require between 140,000 and 190,000 more deep analytical talent positions.

With the predicted job demand explosion, many educational institutions started offering programs related to data science, as example cited by Donoho [43] was the announcement made by the University of Michigan in 2015 about the investment of 100 million dollars in a Data Science Initiative that ultimately hired 35 new faculty.

Another discussion being held during this process is the about the difference between a data analyst and a data scientist. Again, there are many opinions saying that the two roles are essentially the same. Nevertheless, there are efforts being made in order to differentiate them, for example Udacity, one of the most important online courses educational organizations in an official blog entry [93] expresses that a data analyst is essentially a junior data scientist without the mathematical and research background to invent new algorithms but with a strong understanding of how to use existing tools to solve problems.

Now, returning to the second definition of analytics given by Rose [136]: "analytics is the same as data science" and relating it with the eras defined by Davenport [36] it is possible to say that data science and data scientists are for analytics 2.0 what business intelligence and data analysts were for analytics 1.0. They are the respectively the activity and the role that performed analytics in the two eras, and as it seems, the analytics 3.0 era will not bring a change on these terms.

With respect to the relationship between data science and big data, it is difficult to find a publication that does not consider both terms. They both have grown tied to each other from the very beginning, as can be observed for example in the characteristics of a data scientist mentioned in Patil and Davenport [123]: "a new type of professional with the training and curiosity to make discoveries in the world of big data". Nevertheless, many authors such as Donoho [43] and Dhar [40] consider that the fact that the big era brings access to enormous amounts of data does not justify the need of a new term (data science). In other words, even when a data scientist is expected to have the skills to handle such amounts of data and to draw valuable conclusions from them, this is not the main factor the differentiates their role. This is reinforced by (i) the facts that even in the big data era, not all valuable data in an organization will be big data, data scientists will still need to analyze also smaller datasets and (ii) the fact that the required skills to handle big data are still evolving, new tools and frameworks are being developed and what were considered de facto standards in big data such as Apache Hadoop are now being displaced (to some extend) by others such as Apache Spark [162].

At this point, one important question that remains open is: if it its not big data, what then makes the difference between data science and statistics? The answer is not an specific term or skill but the combination of all of them that enables data scientists to effectively dig and extract valuable information from data (big or not). Donoho [43] based on the work of Cleveland [28] and Chambers [23] lists the 6 divisions that should conform data science from an academic point of view:

1. *Data Exploration and Preparation*. Activities involved here are: sanity-checking the data, expose and address unexpected features and anomalies, reformatting, recoding and all the activities involved in preprocessing the data.

2. *Data Representation and Transformation*. The data scientist must be acquainted with the many formats the data can have and the steps needed to transform these according to his needs. This implies the knowledge of different types of structures where the data can be stored, from plain text files to SQL and NoSQL databases and data streams.

3. *Computing with Data*. A data scientist must be able to efficiently develop programs in several languages for data analysis and processing. Including specific computing frameworks for managing complex computational pipelines that may be executed in a distributed

manner in local or cloud computer clusters. Data scientists are also able to develop packages that abstract common pieces of workflow making them available for future projects.

4. *Data Modeling*. This includes generative modeling, in which one proposes a stochastic model that could have generated the data and predictive modeling, in which one constructs methods that make predictions over some given data universe.

5. *Data Visualization and Presentation*. A data scientist must be able to create common plots from the data such as histograms, scatterplots, time series, etc. but also in some occasions he will need to develop custom plots in order to express in a visual manner the properties of a dataset.

6. *Science about Data Science*. This refers to the scientific reflexion about the processes and activities a data scientist performs, for example science about data science happens when they measure the effectiveness of standard workflows in terms of human time, computing resource, results validity or any other performance metric.

3.2.4 Data Science, Data Mining and Machine Learning

The terms data mining and machine learning were not included in the list of terms with difficult definitions at the beginning of this section. However, this subsection is included here with the aim of establishing their relationship with data science.

As it was said in Chapter 2, data mining is defined as the core step of the knowledge discovery in databases process, in this step, different techniques from statistics, probability, databases theory and machine learning are used in order to discover interesting patterns and structures in the data. This definition fits nicely in the data science scheme presented before, since data mining can be placed under the division of data modeling and under the data exploration division given that some data mining techniques are mostly used for this purposes.

On the other hand, machine learning, whose emphasis is on the creation of predictive models from data, also fits well in the data modeling division becoming the very hearth of the whole data science activities. Moreover, a sub area of machine learning know as deep learning, has recently got much attention due to its potential to automatically create complex features by training on large datasets [118]. Thereby, machine learning can also be involved in the data representation division of data science.

Finally, with the aim of wrapping up, Figure 3.2 represents the most important relations between the terms discussed in this section.

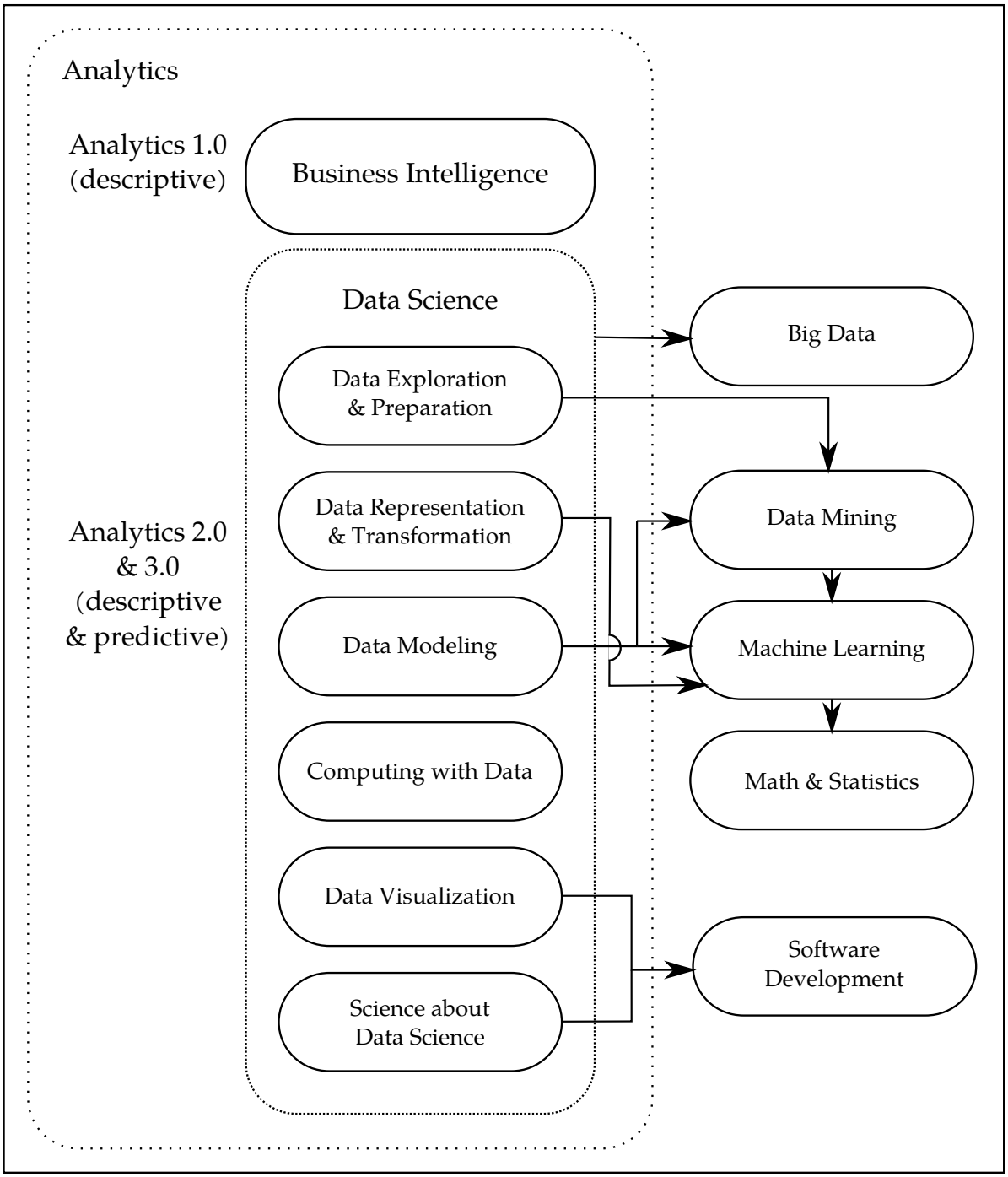

Figure 3.2: Relationships between the discussed terms, the arrow can be interpreted as a “makes use of” relation

CHAPTER 4

DISTRIBUTED SYSTEMS: MAPREDUCE AND APACHE SPARK

The following chapter constitutes the final part of the theoretic foundation of this work, it starts defining what *distributed systems* are and giving their main characteristics. After, the discussion is oriented towards two important libraries and programming models for implementing distributed computations in big data: *MapReduce* and *Apache Spark*. It is worth to mention, that all the algorithms in this work have been implemented using the latter one.

4.1 Distributed Systems

There are many definitions of what a distributed system is, however, to start this section the definition given by van Steen and Tanenbaum [155] was chosen:

> "A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system."

Four important aspects can be remarked from this definition:

1. Autonomy: the elements that conform the system can function in an independent manner without being part of the system.
2. Computing element: Nothing is said about the computing elements being hardware or software, they can be any of them.
3. Single system: the autonomous elements need to collaborate in some form such that they can be seen as a single unit. This collaboration implies that the elements have to communicate between each other, other definitions such as the one given by Coulouris et al. [30] specify that this communication is made through message passing.

4. Users: Similar to the computing elements, nothing is said about the users, they can be persons or software.

Although the term "distributed systems" has never gathered the popularity of the terms discussed in the previous chapter, for example: "big data" and "data science", the reality is that none of these would have the attention they have now without the existence of the current distributed systems. That is, practically all the information that has given rise the current big data era, is coming from distributed systems that have become a normal part of the current society's life, such as: the Internet itself, social networks, email services, search engines, mobile networks, the Internet of Things, etc.

4.1.1 Design Goals

van Steen and Tanenbaum [155] and Coulouris et al. [30] coincide that the main goal of designing a distributed system is to enable or support the sharing of some resource. This resource can be virtually anything from hardware peripherals, sensors, storage, data files, multimedia, software services, etc. Moreover, after having established the resources that are going to be shared and the necessity of a distributed system is clear, there are important goals to follow when designing a distributed system [155], they are listed ahead.

Transparency. In many cases, the distribution of processes and resources wants to be hidden (made transparent) from the final user of the system with the aim of making the use and interaction with the system less complex. The concept of transparency can be applied in several aspects of the system, for example, the access transparency refers to the hiding of the different data representations, file systems, low-level communication protocols, operating systems, etc. Location transparency refers to the fact the user cannot tell where an object is physically located in the system, for example an URL can refer to host anywhere in the planet. Another important aspect to be mentioned is failure transparency, this occurs when partial failures of the system are hidden to the user and the system automatically recovers from them.

Openness. The openness of a system is determined by how easy the system can be extended, used by and integrated into other systems. In order to accomplish this, the system must follow clearly defined standards and their key interfaces must be well documented and published.

Scalability. Bondi [17] defines scalability as "the ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement". Apart from being a design goal, scalability can indeed turn to be the main reason why a distributed system needs to be designed. For example, a centralized system composed by a single computing element can become a bottleneck when the demand outgrows its capacities, thus a scalable distributed system can be the solution.

There are many dimensions where the scalability of a system can be measured, some of them tend to be complex and hard to differentiate [17]. However, Neuman [119] identifies three main dimensions, considered next:

- Size scalability. This is the most direct interpretation, a system is size scalable when it can adequately grow to support more users or provide more shared resources. This type of scalability is basically limited by three factors: the computational capacity of the system, the storage and memory capacities (including size and transfer rate) and the network bandwidth. Faced with these factors, two alternatives can be followed. First, *scaling up* consists in the increasing of the available computing resources of the system nodes that provide the demanded services: upgrading their CPUs, memories, network interfaces, etc. This alternative has an obvious limit when the nodes cannot be improved anymore. The second alternative is known as *scaling out*. In this case, the system is expanded by adding more nodes instead of improving the existent ones. A expanded discussion of this strategy is given ahead.

- Geographical scalability. It is essentially based in network transmission speeds. In a geographically scalable system, the users and resources may lie far apart without having significant communication delay.

- Administrative scalability. It refers to the fact that a system can be administrated by many independent organizations potentially having different policies with respect to the management of resources and security.

Among the three previous dimensions, size scalability is the most interesting for this work. Moreover, it is important to mention that the *scale out* strategy will be the only option when the *scale up* strategy reaches its limit. Also in many cases, scaling out can be much more cost-effective so it can be even used before trying to scale up [114]. van Steen and Tanenbaum [155] mention that there are basically only three techniques to deal with distribution under the scale out strategy: partitioning and distribution of work, hiding communication latencies and replication.

- Partitioning and distribution of work. The most important scaling technique for the purposes of this work consists in spreading the load across nodes. This involves taking a component, dividing it in parts and then distributing those parts between the available servers. For example, this technique is applied in DNS (Internet Domain Name System) were the overall *namespace* is divided into zones and different servers take responsibility of different zones.

- Hiding communication latencies. The most common way of hiding this latencies is by changing the traditional synchronous communications scheme where the requesting application blocks until a response is ready, to an asynchronous scheme where the application makes a request and then it is interrupted when the response is ready. However, not all the applications are able to leverage this type of communication.

- Replication. Replicating components across the system can be helpful in many ways. It helps increasing the availability of a resource: in the case a node with one copy fails, the

other node responds. Caching is a special form of replication, it consists in placing resources more closely to their users, this way latency can be reduced. Moreover, a important design decision is about making the replicas mutable or immutable. If replicas are mutable then synchronization is needed in order to prevent consistency problems. However, in most cases ensuring a strict consistency is very difficult or impossible to implement in a scalable way [155]. For this reason, a distributed database system such as MongoDB provides consistency by reading and writing to a main copy of the data [3], giving the option to the application developer to read from other copies that may not be in a completely consistent state.

4.1.2 Types of Distributed Systems

Before delving into the specific type of distributed system that is of interest in this work, is important to give a glance to the three types of distributed systems considered by van Steen and Tanenbaum [155], namely: distributed computing systems, distributed information systems and pervasive systems.

Distributed computing systems. As its name suggests, this type of distributed systems are focused in realizing computations. This category can be subdivided in three subcategories depending on the homogeneity of the computing nodes and if they are local to an organization or outsourced.

- Cluster computing. As mentioned by Sterling et al. [146], clusters are by far the most common form of supercomputer available. They are conformed by nodes that have similar or identical hardware and software configurations, commonly connected through a high-speed local area network and are usually created as way for scaling out a task. This task is distributed between the nodes and executed in much more efficient manner that a single node in terms of relative-to-cost performance [146]. This last statement is specially true in what are known as *commodity clusters* consisting of affordable and easy to obtain computer hardware.

- Grid computing. According to [105], grid computing can be described by three terms. First of all is *virtualization*, it refers to the fact that grids are conformed by *virtual organizations*, understood as "dynamic groups of organizations that coordinate resource sharing". The next two terms are directly derived from the former: *heterogeneity*, comes from the fact that the virtual organizations may have different computing nodes in terms of hardware, operating systems and network bandwidth. The last term is *dynamic* remarking that organizations in a *virtual organization* can join and leave according to their needs.

- Cloud computing. The USA National Institute of Standards and Technology (NIST) defines cloud computing as a "on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can

be rapidly provisioned and released with minimal management effort or service provider interaction" [112]. Due to its flexibility, cloud computing can be used to create a cluster with homogeneous or heterogeneous nodes than can be managed by a single or by multiple organizations.

Distributed information systems. van Steen and Tanenbaum [155] indicate that this type of distributed systems emerged in organizations that were confronted with a wealth of networked applications, but for which interoperability turned out to be a painful experience. These type of systems are the result of the integration of applications into an enterprise-wide information system.

Pervasive computing systems. This type of systems describe the emerging trend of seamlessly integrating computing into the everyday physical world [143]. Examples of pervasive computing systems are: smart homes, environmental monitoring systems and self-driving cars. Pervasive systems are naturally distributed and have characteristics that make them unique. First, in pervasive systems the separation between the users and the system components is much more blurred. There is often no single dedicated interface, such as a screen/keyboard combination. Instead, they are commonly equipped with many sensors that give input to the system and actuators the provide information and feedback (outputs). Second, pervasive systems are context-aware, they take decisions based on measures such as time, location, temperature, cardiac activity, etc. Another important property in pervasive systems is autonomy. It indicates that this type of systems do not usually have space for a system administrator and have to perform activities such as adding new devices and updating in an automated manner. Finally the last property to mention is intelligence, it suggests that these systems often use methods from the field of artificial intelligence. However, distributed solutions for many problems in the field of artificial intelligence are yet to be found [155].

4.1.3 Parallel Computing

According to Barney [9] *parallel computing* in a simple sense is "the simultaneous use of multiple compute resources to solve a computational problem". This compute resources can include: a single computer with multiple processors, an arbitrary number of computers connected by a network or a combination of both. This broad definition along with the clarification of what a compute resource can be, can lead to observe that a *distributed system* performs parallel computations and that indeed *distributed computing* is a subset of *parallel computing* where the compute resources are autonomous devices. However, there are also more restricted definitions of what *parallel computing* is. For example van Steen and Tanenbaum [155] associates the concept of *parallel processing* to the multiprocessor architecture, where multiple threads of control execute at the same time while having access to a shared data in a *single* computing device. With this definition, *parallel computing* and *distributed computing* are different sets that intersect in the common case where the inner computing devices of a *distributed system* are multiprocessor

devices that perform their work in parallel. Also, under this definition *parallel computing* is very limited in terms of scalability because individual computing devices are often limited in the amounts of memory and processors that can be plugged in the device. This last definition of *parallel computing* is used in this work, unless otherwise specified.

Having presented the generalities of distributed systems, the following two sections delve into two of most important distributed computation libraries used in the big data era: *MapReduce* and *Apache Spark*.

4.2 MapReduce

Considering the classification given in the previous section, *MapReduce* can be defined as a software framework for implementing a distributed computing system. MapReduce was first presented by Dean and Ghemawat [38] from Google Inc. as an important tool used for processing the large amounts of data that this company deals with every day. In many cases, the computations need for this data were conceptually simple and the complexity lied instead in the parallelization of the work, the distribution of the data and the handling of the failures that are common in large commodity clusters. MapReduce addresses this complexities providing the following benefits of a distributed system:

- Distribution transparency in computations. MapReduce provides distribution transparency in computations by automatically parallelizing the execution of a program in a potentially large commodity cluster. Implementing a MapReduce program requires the definition of a *map* function that processes a key/value pair and return a set of intermediate key/value pairs, and a *reduce* function that produces final results by merging all intermediate values associated with the same key. Once this two functions are defined, MapReduce takes care of the work needed for parallelizing and distributing the work by breaking the data into chunks, creating multiple instances of the map and reduce functions, allocating and activating them on available machines in the physical infrastructure, dispatching intermediary results and ensuring optimal performance of the whole system.

- Failure transparency. MapReduce does its best to hide and automatically recover from failures. A master node continuously monitors the execution in the cluster and if some node stops responding or fails, then its work is automatically assigned to another node for completion. The framework warranties that if the user-supplied map and reduce operations are deterministic the results of a successful distributed execution with faults will be the same as if them would have been produced by a non-faulting sequential execution.

- Location transparency. Before MapReduce was published, Ghemawat et al. [58] introduced GFS (Google File System) as a scalable distributed file system for data-intensive applications. According to the authors, at the time of publishing, GFS was widely deployed

within Google and in the largest cluster implementing it, the file system stored hundreds of terabytes across over a thousand machines and was concurrently accessed by hundreds of clients. Similarly to other distributed file systems, GFS provides location transparency by hiding the exact nodes where the data is stored and by automatically moving it for load balance or fault tolerance reasons. It also transparently replicates data for redundancy. GFS was later used as one of the main data sources for the MapReduce engine.

- Size scalability. Size scalability is the main benefit of MapReduce. In order to provide it, MapReduce follows the *scale out* approach, i.e., the computing power of the system can grow by adding more nodes. As mentioned before, MapReduce automatically partitions the data in chunks and assigns it to the different nodes, thus having more nodes will imply that there is less data to process. However, adding more nodes not always will imply an increase of the system performance, other factors can prevent this to happen, being the following the most relevant:
  - Network communication. The amount of nodes that can be added to a network is limited, having too many nodes will saturate the network communications devices affecting the whole system performance.
  - Data is small. When the amount of data that a node receives is below a certain threshold, it can become more efficient to process the whole data using smaller number of nodes, reducing this way the amount of data transmission and the communication needed to coordinate the work.

4.2.1 MapReduce Programming Model

An algorithm that needs to be implemented using MapReduce has to be expressed in terms of two functions: *map* and *reduce*. These function names were taken from functional programming languages such as Lisp even when they were originally not intended to parallelize computation [30]. Every algorithm in MapReduce then will go through the three main steps illustrated in Figure 4.1 and described next.

1. Map step. The *map* function takes as input a key-value pair and produces a set of intermediate key-value pairs. However, the input to the whole algorithm will not only be a single key-value pair but a set of them. The MapReduce engine will distribute these pairs between M cluster tasks known as *mappers* and each of these *mappers* will call the *map* function for each input pair.

2. Shuffle and sort step. After, the set of intermediate key-value pairs returned by all the *mappers* is partitioned into R partitions using their keys, at the same time this partitions are sorted also using the key. This step is performed by R cluster tasks known as *reducers*, one *reducer* per partition.

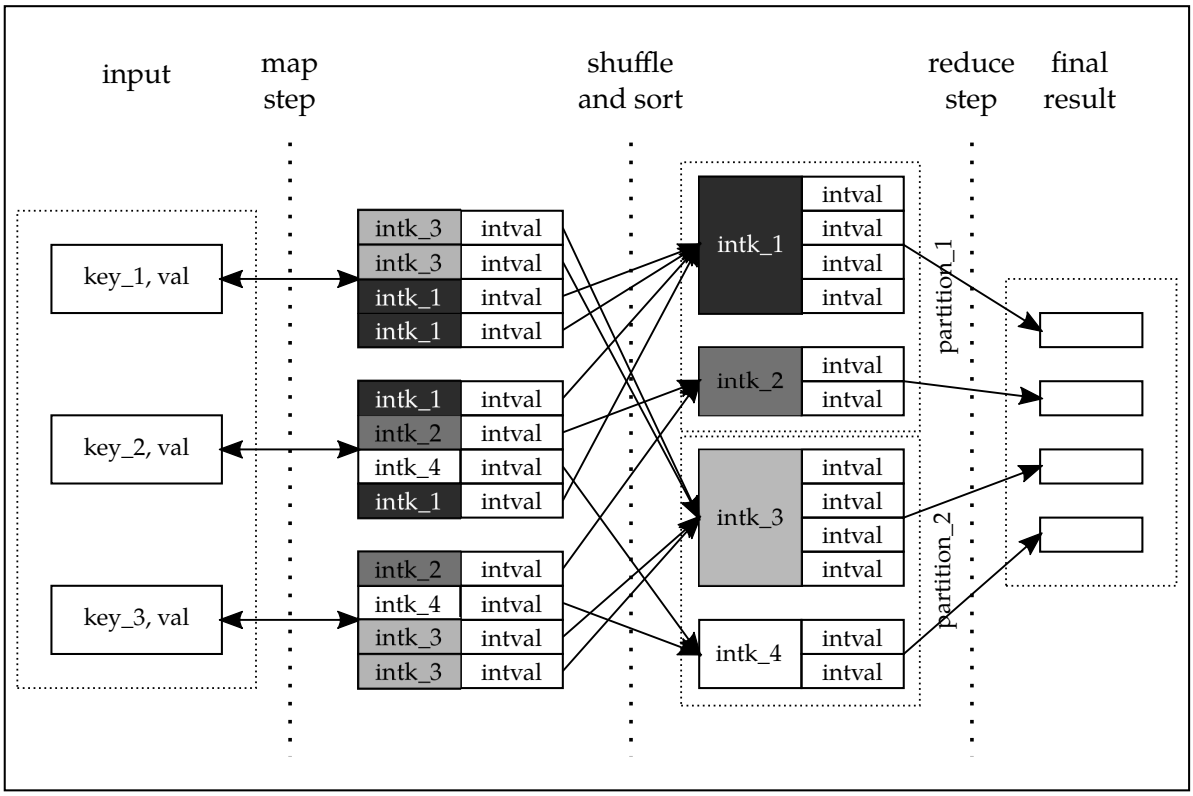

Figure 4.1: Main steps of a MapReduce execution, intk_n and intval, refer to intermediate keys and values respectively

3. Reduce step. The *reduce* function accepts a single intermediate key and a set of values for that key. It merges these values to produce a possibly smaller set of values. In this step, each *reducer* task will invoke the *reduce* function for every single intermediate key and set of values for this key assigned to its partition. The final result will be the join of all the results returned from every invocation to the *reduce* function.

As an illustrative example of a MapReduce implementation, it is possible to consider the problem of creating an inverted index for a set of documents that indicates for every word in the set, the sorted list of documents where they appear. Algorithms 3 and 4 show the pseudo-code to implement this algorithm.

4.3 Apache Hadoop

Apache Hadoop is an open-source software framework for distributed cluster computing and processing of large data sets. The project was co-founded by Doug Cutting and Mike Cafarella, who were inspired by the papers published by Google about the Google File System (GFS) [58]

Algorithm 3 Example of a *map* function implementation

```
1: key : document_id
2: value : document_content
3: words := remove_duplicates(tokenize(value))
4: result := {}
5: for all w in words do
6:     result.add(w, key)
7: end for
8: return result
```

Algorithm 4 Example of a *reduce* function implementation

```
1: key : word
2: values_list : documents_ids
3: return (key, sorted(values_list))
```

and MapReduce [38]. Doug Cutting was a Yahoo employee when the project was published in 2006. The Hadoop project is made up of four main modules:

1. Hadoop Common. This module contains the utilities that give support to the rest of Hadoop modules.

2. Hadoop Distributed File System (HDFS). HDFS is a distributed file system inspired by GFS, it is one the most frequently used technologies for handle large unstructured data [46] and the basis of the whole Hadoop project.

3. Hadoop YARN. YARN is a cluster resource management framework, it takes direct contact with clients who want to make use of a Hadoop cluster, it allocates resources for the applications using a scheduler, monitors jobs and nodes status and deals with failures.

4. Hadoop MapReduce. MapReduce leverages a YARN administered cluster for the processing of large data sets using Google's MapReduce programming model. It is usually fed with data at high rates coming from HDFS.

Being an open-source project, Hadoop became the standard tool for big data processing [97], its name was so frequently mentioned together with big data that it is possible to find phrases in recognized media such as a "Hadoop has been synonymous of big data for years" [101].

As mentioned in [124], at Yahoo, Hadoop became one of the most critical underlying technologies. It was initially applied to web search, but over the years, it became central to many other services with more than one billion users, such content personalization for increasing engagement, ad targeting and optimization for serving the right ad to the right consumer, new revenue streams from native ads and mobile search monetization, mail anti-spam etcetera.

However since its publication in 2006, many tools have appeared with the aim of improving the services offered by Hadoop. Regarding MapReduce, this technology appeared in 2014 under the name of Apache Spark.

4.4 Apache Spark

Apache Spark currently defines itself as "A unified analytics engine for large scale data processing" [1]. It was first published in 2014, originally developed at the University of California, Berkeley's AMPLab and currently maintained by the Apache Software Foundation. Since its publication, Spark has gone through a rapid evolution, changing its own definition in more than one occasion and at the same time becoming one of the most popular frameworks for big data analysis according to the recognized site KDnuggets [1].

Spark and MapReduce are similar in many ways. First of all, Spark is also a distributed computing framework, and has all the benefits mentioned for MapReduce such as: distribution transparency, automatic failure handling and recovering and size scalability. However, Spark has many advantages over MapReduce, it was designed from the beginning to efficiently handle iterative jobs in memory, such as the ones used by many data mining schemes, since this was one of the main problems of MapReduce. This led to the quick development of a machine learning library known as MLib [113]. Moreover, besides the Spark author's own comparison [162, 163], where Sparks results to be up to 100% faster for running a logistic regression model than Hadoop's MapReduce, others comparisons [141] have shown that Spark is faster than MapReduce in most of the data analysis algorithms tested. Second, any MapReduce program can be directly translated to Spark, i.e., the Spark primitives are a superset of MapReduce and the whole MapReduce model can be completely expressed using the $flatMap$, $groupByKey$ and map operations in Spark.

It is worth noting that other models have already tried to fulfill the lack of efficient iterative job handling of MapReduce. Two of them include HaLoop [20] and Twister [48]. However, even though they support executing iterative MapReduce jobs, automatic data partitioning, and Twister has also the ability to keep it in-memory, they both prevented an interactive data mining and can indeed be considered subsets of Spark functionality. In any case, both projects have become outdated.

Liu et al. [100] compared parallelized versions of a neural network algorithm over Hadoop, HaLoop and Spark, concluding that Spark was the most efficient in all cases.

4.4.1 Spark Programming Model

The main concept behind the Spark model is what is known as the resilient distributed dataset (RDD). Zaharia et al. [162, 163] defined an RDD as a read-only collection of objects, i.e., a dataset partitioned and distributed across the nodes of a cluster. The RDD has the ability to automatically

[1] `www.kdnuggets.com`

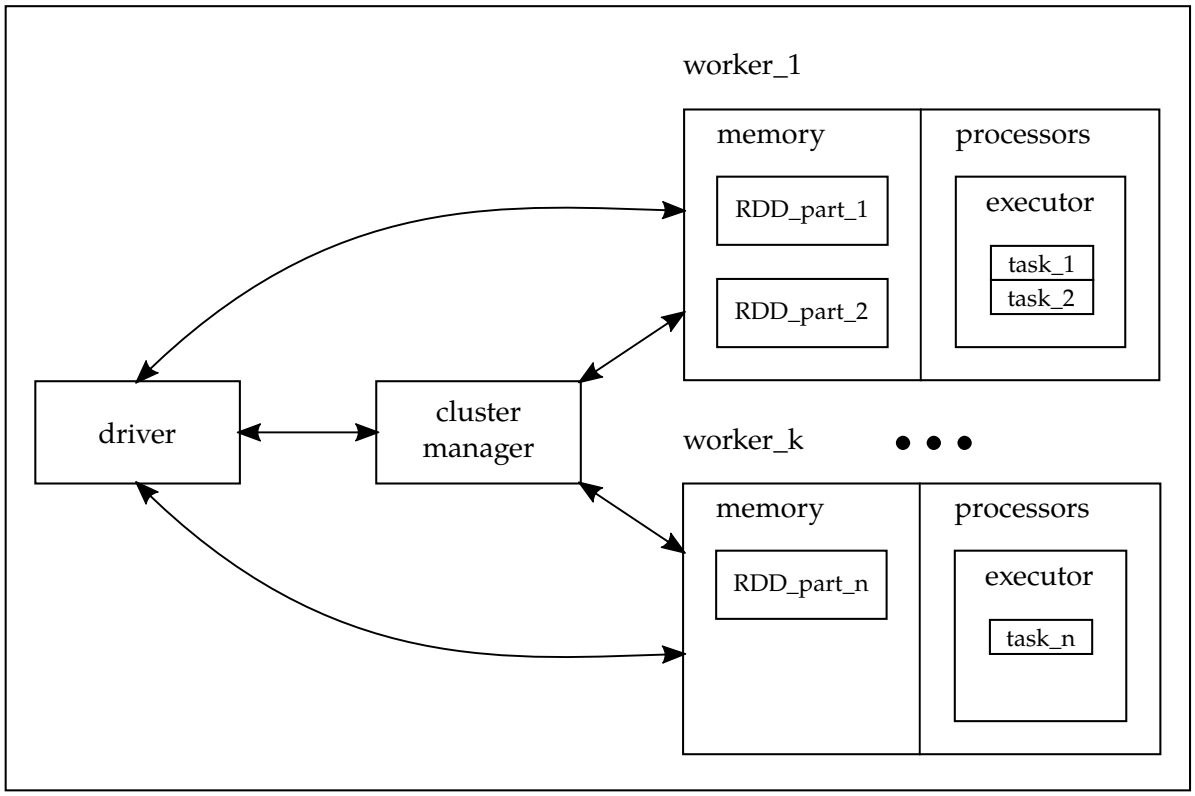

Figure 4.2: Spark Cluster Architecture

recover lost partitions through a lineage record that knows the origin of the data and possible calculations done. Even more relevant for our purposes is the fact that operations run for an RDD are automatically parallelized by the Spark engine; this abstraction frees the programmer from having to deal with threads, locks and all other complexities of traditional parallel programming.

With respect to the cluster architecture, Spark follows the master-slave model. Through a cluster manager (master), a driver program can access the cluster and coordinate the execution of a user application by assigning tasks to the executors, i.e., programs that run in worker nodes (slaves). By default, only one executor is run per worker. Regarding the data, RDD partitions are distributed across the worker nodes, and the number of tasks launched by the driver for each executor is set according to the number of RDD partitions residing in the worker. A detailed view of the discussed architecture with respect to the physical nodes can be observed in Figure 4.2.

Two types of operations can be executed on an RDD, namely, actions and transformations. *Actions* are the mechanism that permit to obtain results from a Spark cluster; five commonly used actions are: $reduce$, sum, $aggregate$, $takeSample$ and $collect$.

The $reduce$ action is used to aggregate the elements of an RDD, by applying a commutative and associative function that receives as arguments two elements of the RDD an returns one element of the same type. Action sum is simply a shorthand for a reduce action that sums all the

elements on the RDD.

The next mentioned action $aggregate$, has a similar behavior to $reduce$, but its return type can be different from the type of the elements of the RDD. It works in two steps: the first one aggregates the elements of each partition and returns an aggregated value for each of them, the second one, merges these values between all partitions to a single one, that becomes the definitive result of the action.

Lastly, actions $takeSample$ and $collect$ are also similar. The former takes an amount of elements and returns a random sample of this size from the RDD. And the latter: $collect$, returns an array with all the elements in the RDD. This operation has to be done with care, to avoid exceeding the maximum memory available to the driver.

Transformations are mechanisms for creating an RDD from another RDD. Since RDDs are read-only, a transformation creating a new RDD does not affect the original RDD. Some of the most important transformations are: map, $flatMap$, $reduceByKey$, $filter$ and $mapPartitions$.

The first two transformations: map and $flatMap$ are similar. Both return a new RDD that is the result of applying a function to each element of the original one. In the case of map, the function applied takes a single argument and returns a single element, thus the new RDD has the same number of elements that the original one. In the case of $flatMap$, the applied function takes a single element but it can return zero or more elements, therefore the resulting RDD is not required to have the same number of elements as the original one.

The next transformation is $reduceByKey$, it can only be applied to what is known as a $PairRDD$, which is an RDD whose elements are key-value pairs, where the keys do not have to be unique. The $reduceByKey$ transformation is used to aggregate the elements of an RDD, which it does by applying a commutative and associative function that receives two values of the PairRDD as arguments and returns one element of the same type. This reduction is applied by key, i.e., elements with the same key are reduced such that the final result is a PairRDD with unique keys, whose corresponding values are the result of the reduction.

The following transformation is $mapPartitions$, which receives, as a parameter, a function that can handle all the elements of a partition and returns another collection of elements to conform a new partition. The $mapPartitions$ transformation is applied to all partitions in the RDD to obtain a new transformed RDD. Since received and returned partitions do not need to match in size, $mapPartitions$ can thus reduce or increase the overall size of an RDD.

Finally, $filter$ transformation is straightforward, it receives a boolean function to discriminate RDD elements to return a subset of it.

CHAPTER 5

State of the Art of Distributed Feature Selection

This chapter represents the end of the first part of this work. Most of the theoretical concepts and technologies needed to develop this and the following chapters were presented in this first part. However, there is still one topic missing, it can be considered as the main link between both parts of this work, this topic is *distributed feature selection*, the location of its discussion at the core of this work coincides with the fact the this is main topic of it. This chapter starts defining what distributed feature selection is and then discusses the recent scientific work related to it.

5.1 Distributed Feature Selection

As its name suggests, distributed feature selection refers to the act of performing feature selection using distributed computations, executed of course, in a distributed system. As it was said in Chapter 2, feature selection is a dimensionality reduction technique that has emerged as an important step in data mining. Its importance is clearly observed in the current era of big data, where the volume, velocity and variety of the data keep increasing together with the necessity of analyzing it to produce value.

Very few works have been devoted to the literal topic of "distributed feature selection" [50, 165], being the work of Bolón-Canedo et al. [12] [14–16] maybe the most notable. Most of the related work in the area prefer to use the term "parallel feature selection" even when many of them implement solutions using distributed computing frameworks such as MapReduce and Spark. This difference in the use of terms is caused by the diverse conceptions of parallel computing discussed in Section 4.1.3 where some consider it as a super set of what distributed computing is and others consider they refer to different sets that can intersect. An example of the latter

conception is the work of Hodge et al. [76] that includes both terms: parallel and distributed on its title and presents two versions of the algorithm: a parallel version for a single machine with a multi-core CPU and a distributed version implemented in a Hadoop cluster.

There are basically two main reasons to perform feature selection in a distributed manner. The first and most common reason is to perform it as an answer to the *problem of big data* (Section 3.1) in the field of feature selection. Although, feature selection algorithms are designed to help data mining techniques deal with the the curse of dimensionality [10], the fact is that under the big data scheme even the feature selection algorithms themselves can have problems to deal with it. Bolón-Canedo et al. [13], in a review of the most widely used feature selection methods, conclude that there is a growing need for scalable and efficient feature selection methods, given that the existing ones are likely to prove inadequate for handling the increasing number of features encountered in big data. This is of course, the reason to perform distributed feature selection followed in this work. The second reason to perform it occurs when the features are inherently distributed in different locations and cannot be centralized, maybe because they belong to different organizations that have they own privacy constraints as in [8] or because the data is distributed over too many distant locations and it becomes impractical to centralize it [32].

5.1.1 Recent Work

Following the reasons explained in the previous section, in recent years, many attempts have been made to achieve more scalable feature selection methods by distributing or parallelizing work. In what follows, a discussion of the recent work on the design of these new methods is presented and organized according to three approaches: (i) search-oriented, (ii) dataset-split-oriented, and (iii) filter-oriented.

Search-oriented parallelizations account for most approaches, in that the main aspects to be parallelized are (i) the search guided by a classifier and (ii) the corresponding evaluation of the resulting models. The following studies are classified in this category:

- Kubica et al. [89] developed parallel versions of three forward-search-based feature selection algorithms, where a wrapper with a logistic regression classifier is used to guide a search parallelized using the MapReduce model.

- Garcia et al. [55] presented a simple approach for parallel feature selection, based on selecting random feature subsets and evaluating them in parallel using a classifier. In their experiments they used a support vector machine (SVM) classifier and, in comparing their results with those for a traditional wrapper approach, found lower accuracies but also much shorter computation times.

- Wang et al. [157] used the Spark computing model to implement a feature selection strategy for classifying network traffic. They first implemented an initial feature selection using

the Fisher score filter [45] and then performed, using a wrapper approach, a distributed forward search over the best m features selected. Since the Fisher filter was used, however, only numerical features could be handled.

- Silva et al. [142] addressed the feature selection scaling problem using an asynchronous search approach, given that synchronous search, as commonly performed, can lead to efficiency losses due to the inactivity of some processors waiting for other processors to end their tasks. In their tests, they first obtained an initial reduction using a mutual information [125] filter and then evaluated subsets using a random forest [75] classifier. However, as stated by those authors, any other approach could be used for subset evaluation.

Dataset-split-oriented approaches have the main characteristic that parallelization is performed by splitting the dataset vertically or horizontally, then applying existing algorithms to the parts and finally merging the results following certain criteria. The following studies are classified in this category:

- Peralta et al. [126] used the MapReduce model to implement a wrapper-based evolutionary search feature selection method. The dataset was split by instances and the feature selection method was applied to each resulting subset. Simple majority voting was used as a reduction step for the selected features and the final subset of feature was selected according to a user-defined threshold. All tests were carried out using the EPSILON dataset also used in the second part of this work.

- Bolón-Canedo et al. [12] proposed a framework to deal with high dimensionality data by first optionally ranking features using a feature selection filter, then partitioning vertically by dividing the data according to features (columns) rather than, as commonly done, according to instances (rows). After partitioning, another feature selection filter is applied to each partition, and finally, a merging procedure guided by a classifier obtains a single set of features. The authors experiment with five commonly used feature selection filters for the partitions, namely, CFS [71], Consistency [33], INTERACT [166], Information Gain [129] and ReliefF [88], and with four classifiers for the final merging, namely, C4.5 [130], Naive Bayes [134], k-Nearest Neighbors [4] and SVM [156], and show that their own approach significantly reduces execution times while maintaining and, in some cases, even improving accuracy.

Finally, as it was mentioned in Chapter 2, filters are one type of feature selection methods that solely rely in the intrinsic properties of the data and frequently require much less computation resources when compared to other types. For these reason, they become an attractive choice when trying to process large amounts of data. However, filter-based feature selection algorithms have asymptotic complexities that depend on the number of features and/or instances in a dataset. Many algorithms, such as the Correlation-based Feature Selection [70, 71] have quadratic

complexities, while the most frequently used algorithms have at least linear complexities [13], this causes that under the big data scheme even this type of algorithms can turn unusable. The work included in the last category: *filter-oriented* methods, include redesigned or new filter methods that are, or become, inherently parallel. Unlike the methods in the other categories, parallelization and distribution in this category methods can be viewed as an internal, rather than external, element of the algorithm.

- Zhao et al. [167] described a distributed parallel feature selection method based on a variance preservation criterion using the proprietary software SAS High-Performance Analytics. [1] One remarkable characteristic of the method is its support not only for supervised feature selection, but also for unsupervised feature selection where no label information is available. Their experiments were carried out with datasets with both high dimensionality and a high number of instances.

- Ramírez-Gallego et al. [132] described scalable versions of the popular mRMR [125] feature selection filter that included a distributed version using Spark. The authors showed that their version that leveraged the power of a cluster of computers could perform much faster than the original and processed much larger datasets.

- Lastly, Eiras-Franco et al. [47], using four distributed feature selection algorithms, three of them filters, namely, InfoGain [129], ReliefF [88] and the CFS [71], reduce execution times with respect to the original versions. However, in the CFS case, the version of those authors focuses on regression problems where all the features, including the class label, are numerical, with correlations calculated using the Pearson coefficient. A completely different approach is required to design a parallel version for classification problems where correlations are based on the information theory. Regarding the ReliefF implementation, even when the processing is distributed in a Spark cluster, it was designed and tested to work only with datasets that fit in the memory of a single computer since it requires to copy the whole dataset on each computer of the cluster. Thereby, their ReliefF implementation has scalability issues and is not able to fully process large datasets as the described in Chapter 1.

The contributions of this dissertation described in the next part of this thesis can be categorized as a *filter-oriented* approach that builds on the two latter works [132], [47]. The fact that their focus was not only on designing an efficient and scalable feature selection algorithm, but also on preserving the original behavior (and obtaining the same final results) of traditional filters, means that previous research focused on those filters is also valid for the new redesigned versions. It is worth mentioning that scalable filters could feasibly be included in any of the

[1] `http://www.sas.com/en_us/software/high-performance-analytics.html`

methods mentioned in the *search-oriented* and *dataset-split-oriented* categories, where an initial filtering step is implemented to improve performance.

5.2 Recent Work on ReliefF and CFS filters

In this last section, background work about the two chosen filter algorithms is presented, considering any type of effort in improving them in any form, in addition to their efficiency and scalability.

5.2.1 Recent Work on ReliefF

As mentioned before, ReliefF is itself an extension of the original Relief algorithm developed by Kira and Rendell [86], the latter was initially limited to binary class problems while the former can handle multi-class problems. According to Urbanowicz et al. [152] Relief is recognized as one of the most prominent filter-base feature selection technique and has given birth to whole family of algorithms denominated as *Relief-based algorithms (RBA)*, being ReliefF probably the most popular.

Urbanowicz et al. [152] make an excellent revision and organization of the recent research about RBAs, they classify the contributions under four categories according to the type of improvement they focus on. Next, these categories are presented together with the considered contributions belonging to them.

Core algorithm improvements. These improvements are related to the core design decisions on the RBA, such as the process to select neighbors, the number of neighbors to select, the weights for the neighbors of same class and the consideration or not of far instances in addition to neighbors. Iterative Relief [44], I-RELIEF [148] and SWRF* [147] are examples of this, they put weights on instances according to their distance to the target instance. The original ReliefF algorithm defines a parameter k that sets the number of neighbors to be used, however other subsequent proposals such as SURF [63] and Iterative Relief [44] employ a distance threshold T to define instances as neighbors. SURF* [64] is an expansion of SURF that considers both nearest and farthest instances from the target instance, I-RELIEF also makes this consideration. MultiSURF* [62] also considers and scores near and far instances, however it defines a dead-band zone in the middle in a manner that instances that fall within this band are excluded from scoring. MultiSURF [153] preserves most aspects of MultiSURF* but eliminates the scoring of far instances because even when this improves the detection of most complex interactions in the genetic analysis field it deteriorates the ability to detect simpler associations. Finally, ReliefSeq [111] takes a different approach from the others, instead of defining a threshold for neighbors selection it dynamically increments k to a k_{max} value in a feature wise manner and selects a k value that produces the largest feature weight in the final scoring. This is of course

very computationally intensive but the authors claim it provides greater flexibility in detecting interactions between features.

Iterative improvements. Sun [148] point out that Relief makes an implicit assumption that the nearest neighbors found in the original feature space are the ones in the weighted space and that the algorithm lacks a mechanism to deal with outlier data. This causes its performance to degrade as the number of irrelevant features increases. The central idea for reducing this problem consists in executing the algorithm many times, each time considering the weights produced in the previous execution such that low scoring features have less influence on the calculation of distances in the following execution. Iterative Relief [44], I-RELIEF [148] and TuRF [117] implement iterative approaches with differences.

Efficiency improvements. Respecting to efficiency, even when ReliefF can be considered fast in normal sized datasets, it has a linear time complexity with respect to the number of features and a scales quadratically with the number if instances when the sample size is set equal to the number of instances ($m = n$). Examples of improvements in this area are the already discussed work of Eiras-Franco et al. [47] and VLSReliefF [49, 94] that works by applying ReliefF to random features subsets an then merging the results by selecting the largest local weight found for each feature.

Data type improvements. One of ReliefF's mayor flaws is its incapacity to detect redundant features, some attempts have been made to overcome this flaw. For example, Li et al. [96] used a forward selection algorithm to select non redundant critical quality characteristics of complex industrial products. Zhang et al. [164] combined ReliefF with mRMR [125] to select non redundant gene subsets that best discriminate biological samples of different types. Other popular area of improvement is the adaptation of Relief to multi-label problems, Reyes et al. [133] presented three proposals about it. Additionally Zafra et al. [161] extended ReliefF to the problem of multi-instance learning.

5.2.2 Recent Work on CFS

The work related to the CFS algorithm has gone down a different path than ReliefF, there is no CFS family of algorithms, since there has been very little work of creating improved versions of it. CFS relevance comes from its practical value, since it has been and continuous to be used in all sorts of applied research [7, 18, 21, 42, 79, 106] from the detection of partial discharges in electrical equipment to the prediction of risk in the life insurance industry.

Section 2.4.4.7 stated the four main elements of CFS, namely: (i) subset evaluation heuristic from test theory, (ii) features correlation measure: symmetrical uncertainty, (iii) search strategy: best first search by default and (iv) secondary heuristic for detection of locally predictive features. Regarding the third element, its important to mention that even when the default search strategy is best first search, the main implementation of CFS in WEKA software has the option to use other strategies such as: genetic algorithm, exhaustive search and random search between others.

This abundance of search options has limited the appearance of CFS versions with modified search options. However, there are examples of this type of adaptation such as the following:

- Soliman and Rassem [145] present a modified version of CFS that uses an hybrid search algorithm named QVICA-with EDA (Quantum Vaccined Immune Clonal Algorithm with Estimation of Distribution Algorithm), this search algorithm implements a mixture of many concepts from different optimization techniques. First of all, CSA (Clonal Selection Algorithm) [37] is an evolutionary algorithm inspired in the behavior of the human immune system, QEA (Quantum Inspired Evolutionary Algorithm) [72] is another evolutionary approach this inspired by the concepts of quantum computing. QICA (Quantum Inspired Immune Clonal Algorithm) [81] is hybrid algorithm that takes concepts form both QIEA and CSA, it was designed in order to reduce population sizes and deal with problems with a higher numbers of dimensions. On the other hand, EDAs (Estimation of Distribution Algorithms) [92] are another type of optimization algorithms with a theoretical foundation on probability theory. They, instead of using crossover or mutation operators like the genetic algorithms, generate new individuals by sampling from a probability distribution estimated from the individuals in a previous generation. Lastly, the QVICA-with EDA algorithm can be described as derived version that introduces the vaccine operator (taken from ICA) and the EDA sampling to the QICA algorithm. The algorithm is compared with a traditional CFS implementation using genetic algorithms and shows to be more effective than it in obtaining relevant and non-redundant features.

- Singh and Singh [144] propose another modification of CFS, named CFS-PSO where a Particle Swarm Optimization (PSO) [84] algorithm is used for search subsets, however their work is limited mainly because no comparisons are made with the original CFS algorithm.

- Dash and Patra [34], in a similar fashion to the previous, present a CFS version named CFS-PSO-QR that uses the PSO algorithm for searching in high dimensional spaces. The authors also propose a small change in the subset evaluation heuristic were the denominator does not refer to the average correlation between the features but to the maximum correlation between them. The “QR” part in the name of the algorithm refers to a post-processing step were a “Quick Reduction” rough set theory based technique is used to find minimal sets of non-redundant features on the results of the CFS-PSO step. The experiments made by the authors are limited since no comparison with the traditional CFS versions are presented.

The other type of contributions related to the CFS algorithm go beyond the modification of the search strategy, the following works are listed under this category:

- Nguyen et al. [120] represent the CFS’s subset evaluation heuristic as an optimization problem and then transform it into a polynomial mixed 0-1 fractional programming problem solving it with the aid of a branch-and-bound algorithm. Their experiments show that the

proposed method outperforms CFS with both the best first or the genetic algorithm search strategies in the selected dataset by removing much more redundant features and still keeping the classification accuracies or even getting better performances.

- Pomsuwan and Freitas [127] adapt the CFS algorithm to the context of longitudinal classification where the features are measured repeatedly across several time points. The proposed adaptation of CFS works in two phases. First, it explicitly treats different values of the same feature across all time points as the same group of temporally related features, performing feature selection separately within each group of such related features. Second, it merges the selected features across all the groups in order to produce a single set of selected features which is then used as input by classification algorithms. In their comparisons with traditional CFS, their method obtained higher predictive accuracy.

Part II

Contribution

CHAPTER 6

DISTRIBUTED FEATURE SELECTION WITH RELIEFF

This chapter describes the first proposal of this doctoral work. A distributed version of the popular ReliefF feature selector named *DiRelieF*. In addition, after presenting and discussing the design of the algorithm, a comparison of the results obtained with DiRelieF and the version implemented in the WEKA platform is presented. This comparison was carried out using four publicly available datasets with numbers of instances in the order of 10^7 and a number of features that ranges in the order of 10^1 to 10^3. All the work proposed here has been published in [122] and the source code is available at GitHub[1].

6.1 DiReliefF

DiReliefF is a distributed and scalable redesign of the original ReliefF algorithm based on the Spark computing model. DiReliefF is able to deal with much larger datasets in terms of both instances and features than the traditional version would be able to handle.

The first design decision is where to concentrate the parallelization effort. As the ReliefF algorithm could be described as an embarrassingly parallel algorithm since its outermost loop goes through completely independent iterations, each of these can be directly executed in different threads as stated by Robnik-Šikonja and Kononenko [135]. However, parallelizing the algorithm in such way ties the parallelization to the number of samples m, and prevents Spark from doing optimizations based on the resources available, the size of the dataset, the number of partitions and the data locality. This would also require that every thread would read through the whole dataset, while as is shown below, there is only one pass needed to process the distances and calculate the feature weights. Furthermore, as mentioned in previous chapters, ReliefF

[1] https://github.com/rauljosepalma/DiReliefF

algorithm's complexity is $\mathcal{O}(m \cdot n \cdot a)$, where n is the number of instances in the dataset, m is the number of samples taken from the n instances and a is the number of features. Moreover, the most complex operation is the selection of the k nearest neighbors for two reasons: first, the distance from the current sample to each of the instances must be calculated with $\mathcal{O}(n \cdot a)$ steps; and second, the selection must be carried out in $\mathcal{O}(k \cdot log(n))$ steps. As a result, the parallelization is focused on these stages rather than on the m independent iterations.

The ReliefF algorithm can be considered as a function applied to a dataset DS, having as input parameters the number of samples m and the number of neighbors k, and returning as output an $a-$size vector of weights W, as shown in (6.1). Thus, the ReliefF algorithm can be interpreted as the calculation of each individual weight $W[A]$, using (6.2), where *sdiffs* (6.3) represents a function that returns the total sum of the differences in the $A-$th feature between a given instance R_i, and a set $NN_{C,i}$ of k neighbors of this instance where all belong to a particular class C. Using this, a series of steps in order to obtain the desired weights vector W can be stated. These steps are summarized in Algorithm 5, shown graphically in Figure 6.1 and are fully described in the following paragraphs.

$$reliefF(DS, s, w) = W \tag{6.1}$$

$$W[A] = \frac{1}{m} \cdot \sum_{i=1}^{m} \left[-sdiffs(A, R_i, cl(R_i)) + \sum_{C \neq cl(R_i)} \left[\left(\frac{P(C)}{1 - P(cl(R_i))} \right) sdiffs(A, R_i, C) \right] \right] \tag{6.2}$$

$$sdiffs(A, R_i, C) = \frac{1}{k} \cdot \sum_{j=1}^{k} diff(A, R_i, NN_{C,i,j}) \tag{6.3}$$

The dataset DS can be defined (see (6.4)) as a set of n instances each represented as a pair $I_i = (F_i, C_i)$ of a vector of features F_i and a class C_i.

$$DS = \{(F_1, C_1), (F_2, C_2), \cdots, (F_n, C_n)\} \tag{6.4}$$

Given the initial definitions and assuming that the features types (nominal or continuous) are stored as metadata, DiReliefF first calculates the maximum and minimum values for all continuous features in the dataset. These values are needed by the *diff* function (see (2.7) and (2.6)) in line 13. DiRelieF, as the original version, uses (2.5) for nominal features and selects between (2.7) and (2.6) for continuous features via an initialization parameter. The task of finding maximum and minimum values is efficiently achieved applying a $reduce$ action with a function $fmax$ ($fmin$) that given two instances returns a third one containing the maximum (minimum) values for each continuous feature. This is shown for maximum values on (6.5).

Algorithm 5 DiReliefF

1: DS = input dataset
2:
3: {Begin steps distributed in cluster}
4:
5: (MAX, MIN) := max and min values for all continuous feats via $reduce$ over DS
6: P := all class priors via $reduceByKey$ over DS
7: R := m samples obtained via $takeSample$ from DS
8: DD := distances RDD from DS to R via map over DS
9: NN = global nearest neighbors matrix via $aggregate$ over DD
10:
11: {End of distributed steps}
12:
13: $SDIF$ = sum of differences matrix using $diff$, MAX and MIN over NN
14: W = weights vector using $SDIF$, P and equation (6.2)
15: **return** W

$$MAX = DS.reduce(fmax)$$

$$MAX = (max(F_1[1], \cdots, F_n[1]), \cdots, max(F_1[a], \cdots, F_n[a])) \tag{6.5}$$

The next step calculates the prior probabilities of all classes in DS. These values can be essentially obtained by the means of a map and a $reduceByKey$ transformation. The former returns a dataset with all instance classes paired with a value of one. The latter sums these ones using the class as a key, thereby obtaining a set of pairs $(C_i, count(C_i))$ containing the classes and the number of instances in DS belonging to that class, which can simply be divided by n to obtain the priors. Equations (6.6) depict the previous discussion. Note that with the use of $collect$, P is turned into a local array rather than an RDD.

$$f(I) = (cl(I), 1)$$

$$g(a,b) = a + b$$

$$h((a,b)) = (a, b/n)$$

$$P = DS.map(f).reduceByKey(g).map(h).collect()$$

$$P = \{(C_1, prior(C_1)), \cdots, (C_c, prior(C_c))\} \tag{6.6}$$

A rather short step is the selection of the m samples, this is accomplished with the use of the $takeSample$ action, as shown in (6.7).

$$R = (R_1, \cdots, R_m) = DS.takeSample(m) \tag{6.7}$$

Next comes the computationally intensive step of finding the k nearest neighbors of each sample for each class. First, the distances from every sample m to each of the n instances must be found. This can be directly accomplished by the means of a map transformation applied to DS, where for every instance I, a vector of distances from it to the m samples is returned (as shown in (6.8)).

$$
\begin{aligned}
distances(I) &= (distance(I,R_1),\cdots,distance(I,R_m)) \\
f(I) &= (I, distances(I)) \\
DD &= DS.map(f) \\
DD &= \{(I_1, distances(I_1)),\cdots(I_n, distances(I_n)))\}
\end{aligned} \tag{6.8}
$$

Next, as shown in (6.9), the nearest neighbors NN matrix is obtained by using an $aggregate$ action, where each element $NN_{C,i}$ of this matrix is a set of the k nearest neighbors of a sample R_i belonging to a class C. The use of an $aggregate$ action is an important design decision since it allows to obtain the NN matrix with a single pass through DD. Previous experiments showed that trying to find the neighbors for each sample in independent loops was much less efficient.

As stated before, the $aggregate$ action has two steps. The first step is defined in the function $localNN$ that returns a local neighbors matrix LNN for each partition of the RDD. This matrix has a similar structure as the NN matrix but each element is treated instead as a k-sized binary heap that is used to incrementally store the nearest neighbors found during the traverse of the local partition.

The second step of the $aggregate$ action is the merging of the local matrices. The defined function $mergeNN$ combines two local matrices by merging its individual binary heaps, keeping only the elements with shorter distances. Once both functions have been defined, a call to the $aggregate$ action can be performed, providing also an empty matrix LNN structure (with empty heaps) so that the $localNN$ can start aggregating neighbors to it. Lastly, since the NN matrix is obtained via an action, it is a local object and not an RDD. This step, which has been fully implemented using parallel and distributed operations, is the most complex step in the algorithm.

$$
\begin{aligned}
NN &= \begin{bmatrix} NN_{1,1} & \cdots & NN_{1,m} \\ \vdots & \ddots & \vdots \\ NN_{c,1} & \cdots & NN_{c,m} \end{bmatrix} \\
NN_{C,i} &= (N_1,\cdots,N_k \mid \forall N\ cl(N) = C) \\
localNN(I, LNN_{in}) &= LNN_{out} \\
mergeNN(LNN_a, LNN_b) &= LNN_{merged} \\
NN &= DS.aggregate(emptyNN, localNN, mergeNN)
\end{aligned} \tag{6.9}
$$

Once the NN matrix is stored on the driver program, operations are not distributed in the cluster anymore. However, this is not a problem but a requirement, because NN matrix is small, $c \times m$. The last step consists in obtaining the matrix $SDIF$. Each element $SDIF_{C,i}$ represents an a-size vector, and each element of this vector stores the sum of the differences for the A-th feature between the $NN_{C,i}$ set of neighbors and the R_i sample. Each element of the vector $SDIF_{C,i}$ can be calculated by mapping the *diff* function over the A-th feature of all the instances in the $NN_{C,i}$ set and then summing these differences. This map and sum functions are not RDD-related anymore, but local equivalents that execute on the driver's local threads. Finally, each element of each vector of the matrix $SDIF$ effectively represents the value shown in (6.3), thus, the final vector of weights, W, can be easily calculated by applying (6.2) using the already obtained P set with the prior probabilities. The above discussion is depicted in 6.10.

$$
\begin{aligned}
SDIF_{C,i} &= (sdiffs(1,R_i,C),\cdots,sdiffs(a,R_i,C)) \\
f(N) &= diff(A,N,R_i) \\
SDIF_{C,i,A} &= sdiffs(A,R_i,C) = NN_{C,i}.map(f).sum/k \\
SDIF &= \begin{bmatrix} SDIF_{1,1} & \cdots & SDIF_{1,m} \\ \vdots & \ddots & \vdots \\ SDIF_{c,1} & \cdots & SDIF_{c,m} \end{bmatrix}
\end{aligned}
$$

(6.10)

Finally, there is one implementation issue worth mentioning. As previously stated, an RDD is designed to be kept in memory but this does not happen automatically, therefore, if the RDD is going to be used in future operations it must be explicitly cached. In our case, the most complex part of the algorithm is the calculation of the DD and NN matrices. This calculation is effectively performed in a single pass through the dataset initiated by the $aggregate$ action, and therefore, caching of any intermediate result would indeed cause a waste of resources. However, the initial part of the algorithm that requires the calculations of the maximum, minimum and priors, each require a pass through the dataset DS and therefore can take advantage of caching but only when it fits in the distributed memory of the cluster. When it does not, caching does not help because its benefits are outweighed by the time needed for writing the dataset to disk. As a result, in our implementation caching is disabled by default but can be enabled with a parameter (it should be enabled only when it can be assured that the dataset fits in the distributed memory).

6.2 Experiments and Results

In this section, experimental results obtained from different executions of the proposed algorithm are presented. The experiments were performed with the aim of testing the algorithm scalability and its time and memory consumption with respect to a traditional version in WEKA. Further

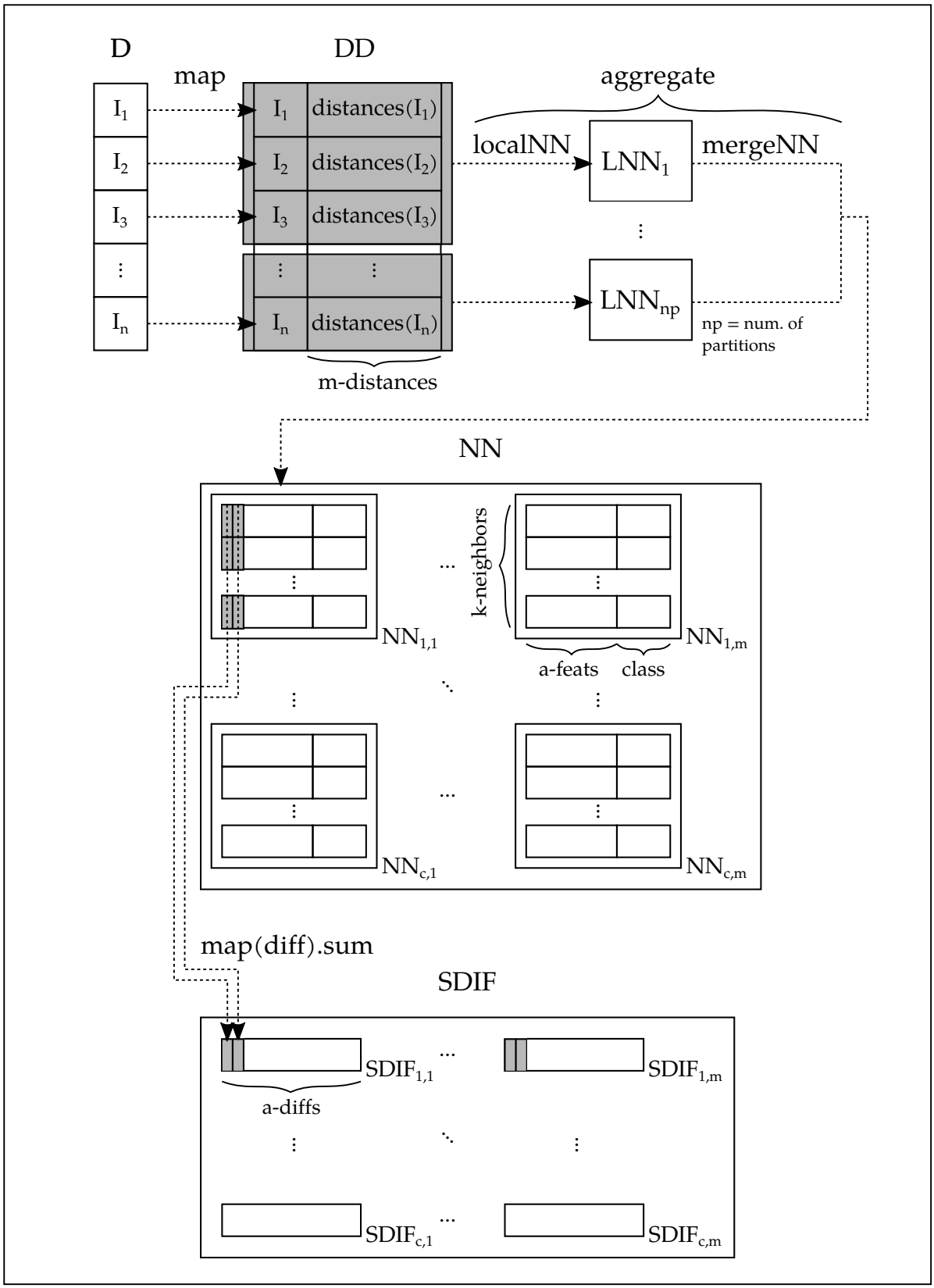

Figure 6.1: DiReliefF's Main Pipeline

tests were also performed in order to observe sample sizes where the algorithm's weights become stable. Since the distributed version was designed to return exactly the same results as the non-distributed one, there was no need to perform tests comparing them.

For the realization of the tests, an 8-node cluster of virtual machines was used, one node is used as a master and the rest are slaves, the cluster runs over the following hardware-software configuration:

- Host Processors: 16 X Dual-Core AMD Opteron 8216
- Host Memory: 128 GB DDR2
- Host Storage: 7 TB SATA 2 Hard Disk
- Host Operating System: CentoOS 7
- Hypervisor: KVM (qemu-kvm 1.5.3)
- Guests Processors: 2
- Guests Memory: 16 GB
- Guests Storage: 500 GB
- Java version: OpenJDK 1.8
- Spark version: 1.6.1
- HDFS version: 2.6.4
- WEKA version: 3.8

During the first part of the experimental work, the ECBDL14 [6] dataset was used. This dataset comes from the Protein Structure Prediction field, and it was used during the ECBLD14 Big Data Competition of the GECCO'2014 international conference. The dataset has approximately 33.6 million instances, 631 attributes, 2 classes, 98% of negative examples and occupies about 56GB of disk space. In a second part of the experimental work, the other three datasets described in Table 6.1 are used. HIGGS [137], from the UCI Machine Learning Repository [98], is a recent dataset that represents a classification problem that distinguishes between a signal process which produces the Higgs bosons and a background process which does not. KDDCUP99 [104] represents data from network connections and classifies them between normal connections and different types of attacks (a multi class problem). Finally, EPSILON is an artificial data set built for the Pascal Large Scale Learning Challenge in 2008[2].

[2] `http://largescale.ml.tu-berlin.de/about/`

Table 6.1: Datasets used in the experiments

Dataset	No. of Inst.	No. of Feats.	Features Types	Problem Type
ECBDL14	~33.6 million	632	Numerical and Categorical	Binary
HIGGS	11 million	28	Numerical	Binary
KDDCUP99	~5 million	42	Numerical and Categorical	Multi-class
EPSILON	1/2 million	2,000	Numerical	Binary

Initially, all tests are run with a number of neighbors $k = 10$ which is a typical choice [88], and a relatively low number of samples $m = 10$ to keep execution times reasonable. However, during the stability tests larger number of samples are used. In addition, HDFS is used to store all the datasets or samples of datasets in the experiments related to the distributed version. Conversely, the local file system is used for the tests with the traditional version of ReliefF.

Regarding the rest of the implementation parameters, two of them were left fixed for all the experiments. The first one selects the original *diff* function in (2.6) for the distance evaluation of numeric features, as this is the only one implemented in the WEKA version. However, it is worth mentioning that since the ramp function in (2.7) requires a constant number of extra operations, selecting it will simply multiply by a constant factor the total number of operations that the algorithm has to execute, while also bringing the benefits mentioned before and explained by Hong [77]. The second parameter refers to whether or not apply caching in Spark. As stated above, caching provides benefits only when the entire dataset fully fits in memory. Therefore, it was decided to disable caching for all the experiments in order to facilitate the interpretation of the results.

An important configuration issue refers to driver memory consumption. In Spark computation model, there is no communication between tasks, so all the task's results will be sent to the driver. This is especially important for the $aggregate$ action, because every task performing the $localNN$ operation will return a LNN matrix to the driver that then is going to be merged. The Spark configuration parameter $spark.driver.maxResultSize$ has a default value of 1GB but it was set to 6GB for all the experiments performed. This is specifically important for tests involving larger matrix sizes, i.e., those with higher values of m or c.

6.2.1 Empirical Complexity

Figure 6.2 shows time and memory consumption behavior of the distributed version of ReliefF version versus the one implemented in the WEKA platform for incrementally sized samples of the ECBDL14 dataset. This dataset was selected because it has the largest amount of instances and allows to show the limits of the WEKA implementation. Since the number of operations

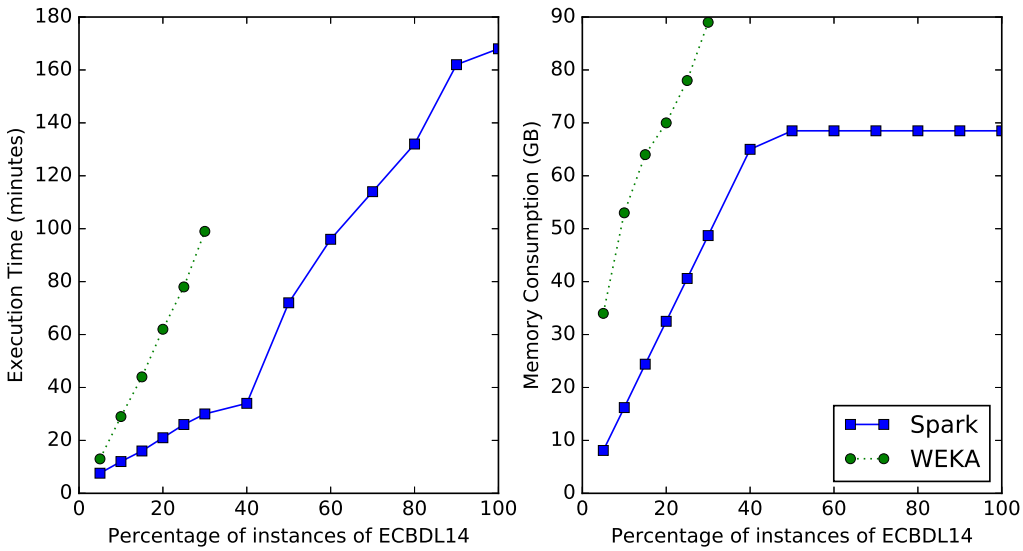

Figure 6.2: Execution time and memory consumption of Spark DiRelieF and WEKA ReliefF versions

performed by the ReliefF algorithm depends only on the parameters m, n and a, and not on the internal characteristics of the data, the other three datasets were not considered for this part. To make the comparison possible, the WEKA version was executed under the host environment with no virtualization. It is worth noting that for the WEKA version a 30% sample was the largest that could be tested because larger samples would need more memory than is available in the system (showing the lack of scalability). The distributed version, in addition to being able to handle the whole dataset, preserves a linear behavior in relation to the number of instances, and it is also capable of processing data in less time by leveraging the cluster nodes. Another fact to observe is the change in the slope of the linear behavior observed by the Spark version between the 40% and 50% samples of the dataset. This is due to the fact that during this interval the dataset overflows the available memory in the cluster and the Spark engine starts using disk storage. In other words, the algorithm is capable of maintaining a linear complexity even when the dataset does not fit into memory.

The mentioned overflow issue can be observed on the right graph in Figure 6.2. This graph shows a previously mentioned advantage of Spark, i.e, it is designed to run in commodity hardware. A simple look to the memory consumption of the traditional version shows that the required amount of memory quickly overgrows beyond the limits of an ordinary computer. However, the distributed version using nodes with 16GB of memory is enough to handle the task.

Analogous run time results are obtained by varying the number of features and the number of samples (see Figure 6.3) confirming an empirical complexity equivalent to the original one, i.e., $\mathcal{O}(m \cdot n \cdot a)$.

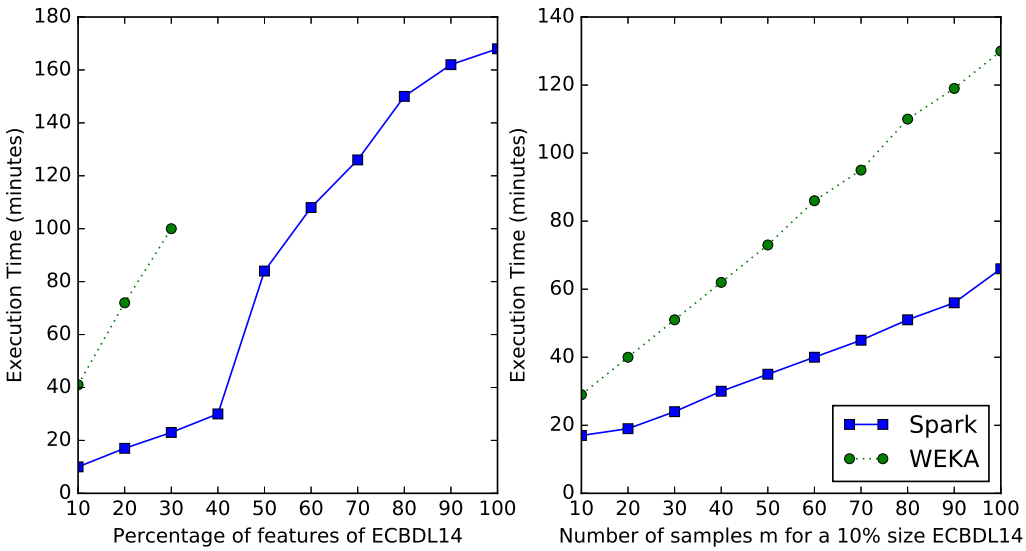

Figure 6.3: Execution time of Spark DiReliefF and WEKA ReliefF with respect to parameters a and m

6.2.2 Scalability

In order to test scalability and to keep execution times within manageable limits, the largest test was performed with a 30% sample size of the ECBDL14 dataset. Following this, 10% and 1% sample sizes were used. Also, in order to examine a smaller dataset, tests with 50% and 10% samples of the HIGGS dataset were run.

Figure 6.4 shows the behavior of the distributed algorithm with respect to the number of cores used. As it can be observed, adding cores at the beginning greatly contributes to reducing the execution time, but once a threshold is reached, the contribution is rather subtle or even null. Such threshold depends on the size of the dataset, in such a way that larger datasets can take further advantage of larger number of cores. On the other hand, smaller datasets will face the case were they do not have enough partitions to be distributed over all the cluster nodes. In this latter case, adding more nodes will not provide any performance improvement. As an example to quantify this fact, it can be observed in Figure 6.4 that the 30% sized sample of the ECBDL14 dataset can take advantage of 7 cores, while the 1% sample size only can take advantage of 4 cores. Similarly, there is no practical advantage of using more than 1 core for the 10% sample size of the HIGGS dataset.

Figure 6.4 also shows with an horizontal line the execution time of the WEKA version for 30% sample of the ECBDL14 dataset. Since it can only take advantage of one core, the execution time is constant. However, the time is better than the distributed version in the case where 4 or less cores are involved. This is because it does not need to deal with the driver scheduling, the selection of an executor and communication between both of them over the network, as the

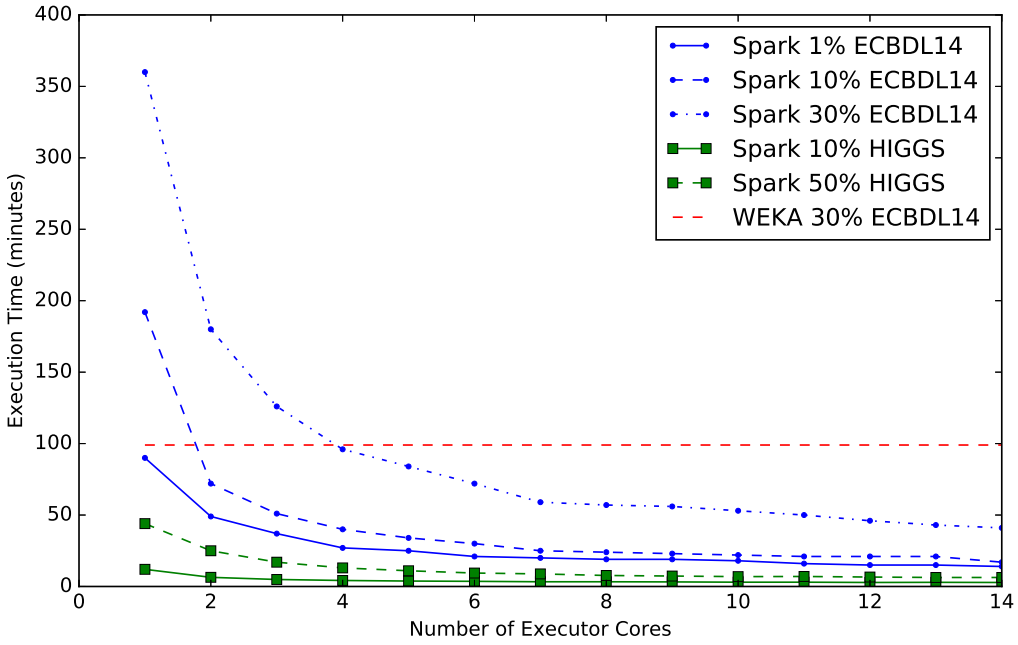

Figure 6.4: Execution time of Spark DiReliefF and WEKA ReliefF with respect to the number of cores involved

distributed version does. For this reason, the distributed version of ReliefF is only useful for large datasets.

6.2.3 Stability

The following set of tests was made in order to check the stability of the algorithm. In this case, with stability refers to similarity of the assigned weights for different executions in the same dataset. The stability is important in the case of ReliefF since it selects a random sample in every execution.

Many stability measures have been defined. A commonly used measure is the Consistency Index presented by Kuncheva [90]. However, it cannot be directly applied because it requires to define a threshold for the selection of a subset. Kalousis et al. [83] proposed the use of the Pearson correlation coefficient to measure the similarity between features rankings, but previous tests showed that it returns unstable results for the EPSILON and HIGGS (pure numerical) datasets. For these reasons, here a simpler stability indicator was chosen, an average of the absolute differences between every feature weights of two rankings. More formally, having two feature ranking vectors $W1$ and $W2$, the average difference between them is calculated as shown in (6.11).

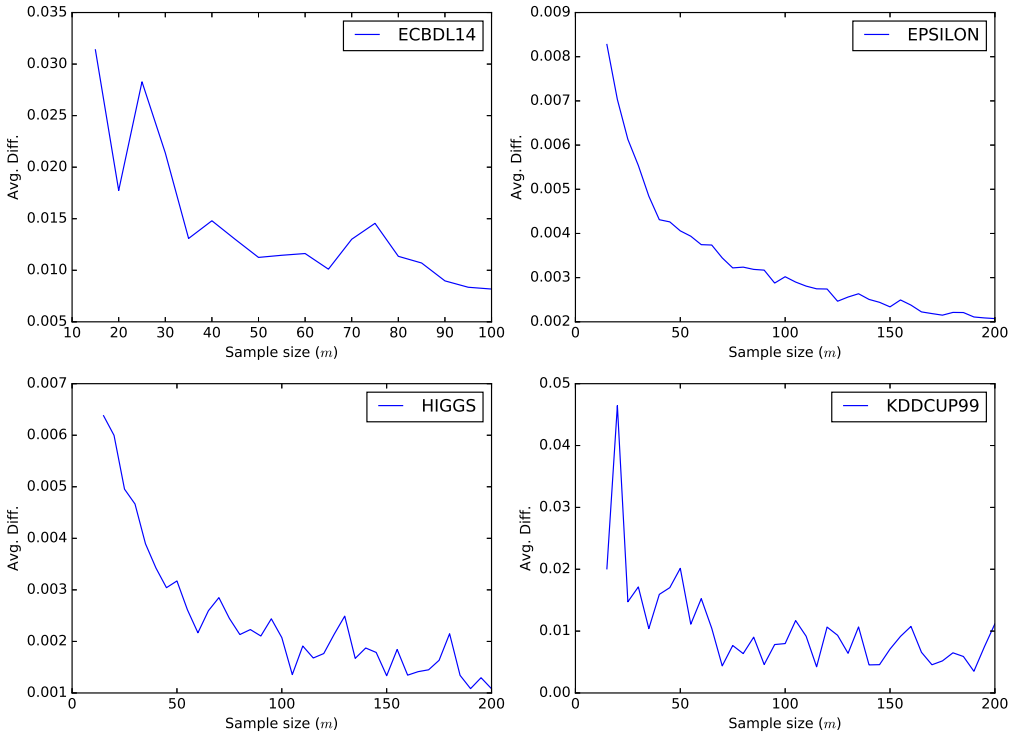

Figure 6.5: DiReliefF's average difference in weight ranks for increasing values of m in different datasets

$$W1 = (w1_1, w1_2, \cdots, w1_a)$$
$$W2 = (w2_1, w2_2, \cdots, w2_a)$$
$$AvgDiff = \frac{1}{a}\sum_{A=1}^{a} |w1_A - w2_A| \tag{6.11}$$

As Robnik-Šikonja and Kononenko [135] stated, the correct sample size m is problem dependent, and as a consequence, all of the datasets described in Table 6.1 were used. Moreover, their experiments showed that as the number of examples increases, the required sample size diminishes. Figure 6.5 shows the results obtained and evidences that the biggest gain in stability is obtained with a sample size between 50 and 100 instances, thereby confirming that relative small sample sizes are enough to obtain stable results in the large scale datasets tested.

CHAPTER 7

DISTRIBUTED FEATURE SELECTION WITH CFS

The chapter ahead presents the second and last contribution of this thesis work. A distributed version of the well known CFS feature selection technique, called *DiCFS*. In fact, two versions of the algorithm were implemented and compared between them and with a baseline, represented by the classical non-distributed implementation of CFS in WEKA. Also, their benefits in terms of reduced execution time are compared with those of the CFS version developed by Eiras-Franco et al. [47] for regression problems. All the source code was published at Github[1].

7.1 Distributed Correlation-Based Feature Selection (DiCFS)

As just mentioned, two algorithms conform the proposal. They represent alternative distributed versions that use different partitioning strategies to process the data. As stated previously, CFS has a time execution complexity of $\mathcal{O}(m^2 \cdot n)$ where m is the number of features and n is the number of instances. This complexity derives from the first step shown in Algorithm 2, the calculation of $\binom{m+1}{2}$ correlations between all pairs of features including the class, and the fact that for each pair, $\mathcal{O}(n)$ operations are needed in order to calculate the entropies. Thus, to develop a scalable version, the main focus in parallelization design must be on the calculation of correlations.

Another important issue is that, although the original study by Hall [71] stated that all correlations had to be calculated before the search, this is only a true requisite when a backward best-first search is performed. In the case of the search shown in Algorithm 2, correlations can be calculated on demand, i.e., on each occasion a new non-evaluated pair of features appears during

[1] `https://github.com/rauljosepalma/DiCFS`

the search. In fact, trying to calculate all correlations in any dataset with a high number of features and instances is prohibitive; the tests performed on the datasets described in Section 7.2 show that a very low percentage of correlations is actually used during the search and also that on-demand correlation calculation is around 100 times faster when the default number of five maximum fails is used.

Next, descriptions of the two alternative methods for calculating these correlations in a distributed manner depending on the type of partitioning used, are given.

7.1.1 Horizontal Partitioning

Horizontal partitioning of the data may be the most natural way to distribute work between the nodes of a cluster. Considering the default layout where the data is represented as a matrix D in which the columns represent the different features and the rows represent the instances, then it is natural to distribute the matrix by assigning different groups of rows to nodes in the cluster. And if this matrix is represented as an RDD, this is exactly what Spark will automatically do.

Once the data is partitioned, Algorithm 2 (omitting line 1) can be started on the driver. The distributed work will be performed on line 8, where the best subset in the queue is expanded and, depending on this subset and the state of the search, a number nc of new pairs of correlations will be required to evaluate the resulting subsets. Thus, the most complex step is the calculation of the corresponding nc contingency tables that will allow to obtain the entropies and conditional entropies that conform the symmetrical uncertainty correlation (see Equation (2.4)). These nc contingency tables are partially calculated locally by the workers following Algorithm 6. As can be observed, the algorithm loops through all the local rows, counting the values of the features contained in *pairs* (declared in line 1) and storing the results in a map holding the feature pairs as keys and the contingency tables as their matching values.

The next step is to merge the contingency tables from all the workers to obtain global results. Since these tables hold simple value counts, they can easily be aggregated by performing an element-wise sum of the corresponding tables. These steps are summarized in Equation (7.1), where $CTables$ is an RDD of keys and values, and where each key corresponds to a feature pair and each value to a contingency table.

Algorithm 6 function localCTables(pairs)(partition)

1: $pairs \leftarrow nc$ pairs of features
2: $rows \leftarrow$ local rows of $partition$
3: $m \leftarrow$ number of columns (features in D)
4: $ctables \leftarrow$ a map from each pair to an empty contingency table
5: **for all** $r \in rows$ **do**
6: **for all** $(x, y) \in pairs$ **do**
7: $ctables(x, y)(r(x), r(y))$ += 1
8: **end for**
9: **end for**
10: **return** $ctables$

$$
\begin{aligned}
pairs &= \{(feat_a, feat_b), \cdots, (feat_x, feat_y)\} \\
nc &= |pairs| \\
CTables &= D.mapPartitions(localCTables(pairs)).reduceByKey(sum) \\
CTables &= \begin{bmatrix} ((feat_a, feat_b), ctable_{a,b}) \\ \vdots \\ ((feat_x, feat_y), ctable_{x,y}) \end{bmatrix}_{nc \times 1}
\end{aligned}
$$

(7.1)

Once the contingency tables have been obtained, the calculation of the entropies and conditional entropies is straightforward since all the information necessary for each calculation is contained in a single row of the $CTables$ RDD. This calculation can therefore be performed in parallel by processing the local rows of this RDD.

Once the distributed calculation of the correlations is complete, control returns to the driver, which continues execution of line 8 in Algorithm 2. As can be observed, the distributed work only happens when new correlations are needed, and this occurs in only two cases: (i) when new pairs of features need to be evaluated during the search, and (ii) at the end of the execution if the user requests the addition of locally predictive features.

To sum up, every iteration in Algorithm 2 expands the current best subset and obtains a group of subsets for evaluation. This evaluation requires a merit, and the merit for each subset is obtained according to Figure 7.1, which illustrates the most important steps in the horizontal partitioning scheme using a case where correlations between features f2 and f1 and between f2 and f3 are calculated in order to evaluate a subset.

7.1.2 Vertical Partitioning

Vertical partitioning has already been proposed in Spark by Ramírez-Gallego et al. [132], using another important FS filter, mRMR. Although mRMR is a ranking algorithm (it does not select

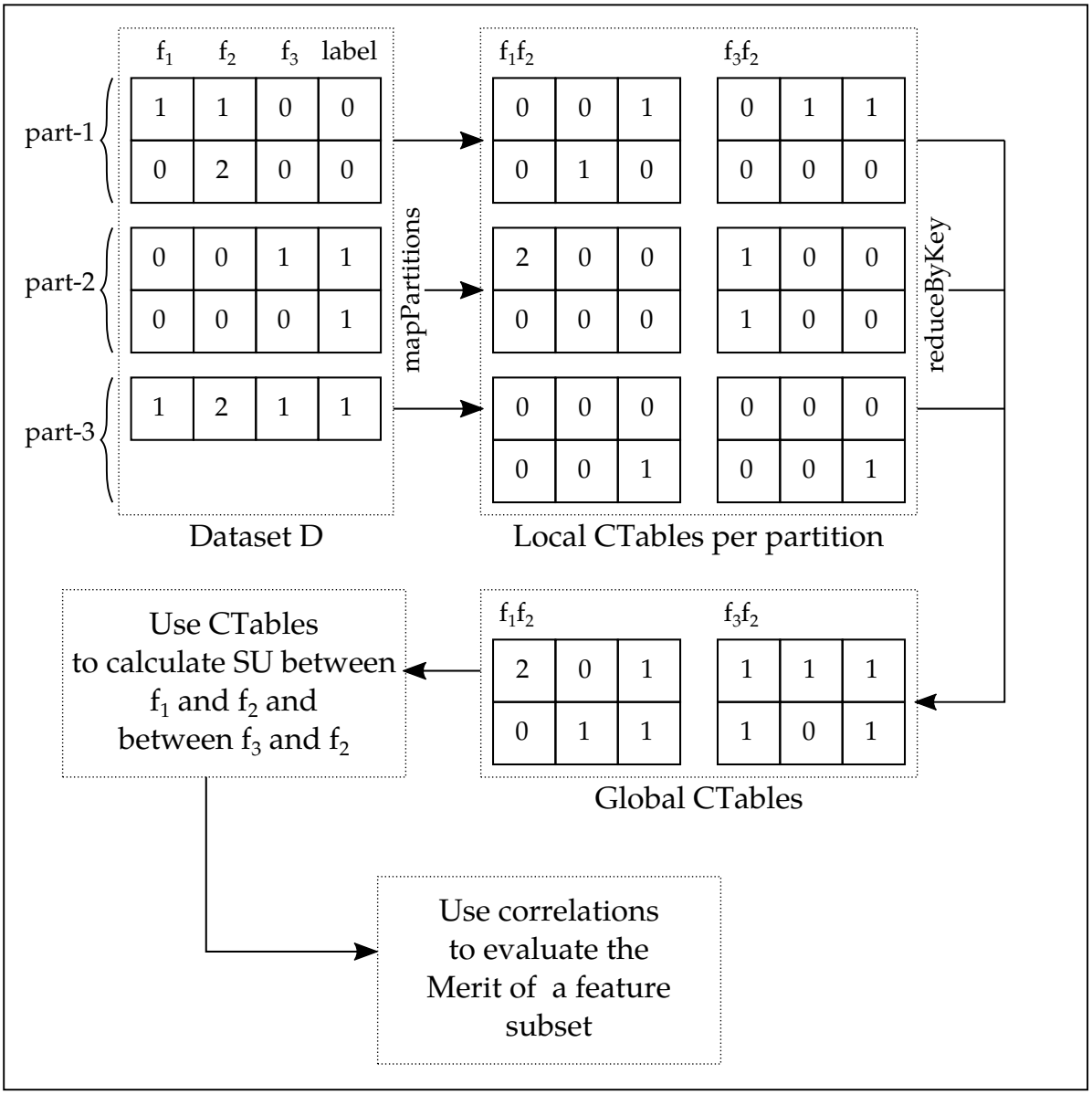

Figure 7.1: Horizontal partitioning steps for a small dataset D to obtain the correlations needed to evaluate a features subset

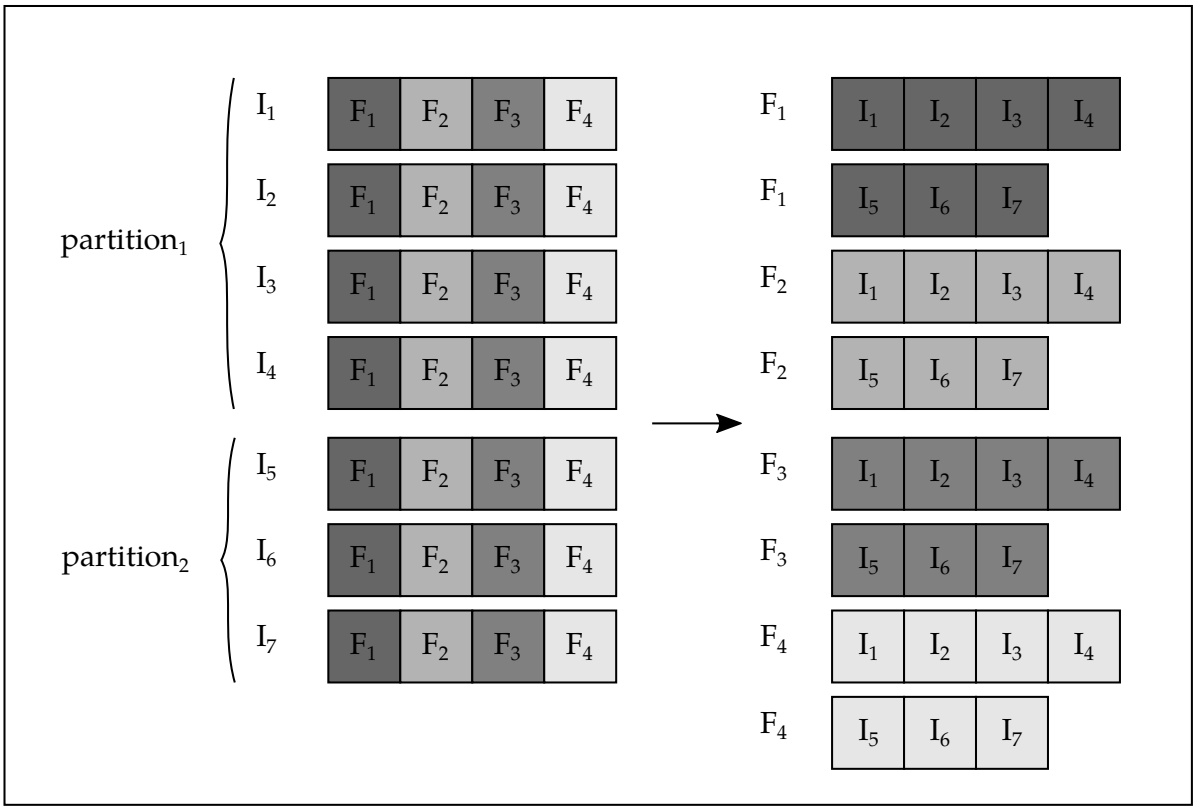

Figure 7.2: Example of a columnar transformation of a small dataset with two partitions, seven instances and four features (from [132])

subsets), it also requires the calculation of information theory measures such as entropies and conditional entropies between features. Since data is distributed horizontally by Spark, those authors propose two main operations to perform the vertical distribution:

- *Columnar transformation*. Rather than use the traditional format whereby the dataset is viewed as a matrix whose columns represent features and rows represent instances, a transposed version is used in which the data represented as an RDD is distributed by features and not by instances, in such a way that the data for a specific feature will in most cases be stored and processed by the same node. Figure 7.2, based on [132], explains the process using an example based on a dataset with two partitions, seven instances and four features.

- *Feature broadcasting*. Because features must be processed in pairs to calculate conditional entropies and because different features can be stored in different nodes, some features are broadcast over the cluster so all nodes can access and evaluate them along with the other stored features.

In the case of the adapted mRMR [132], since every step in the search requires the comparison of a single feature with a group of remaining features, it proves efficient, at each step, to broadcast this single feature (rather than multiple features). In the case of the CFS, the core issue is that, at any point in the search when expansion is performed, if the size of subset being expanded is k, then the correlations between the $m-k$ remaining features and $k-1$ features in the subset being expanded have already been calculated in previous steps; consequently, only the correlations between the most recently added feature and the $m-k$ remaining features are missing. Therefore, the proposed operations can be applied efficiently in the CFS just by broadcasting the most recently added feature.

The disadvantages of vertical partitioning are that (i) it requires an extra processing step to change the original layout of the data and this requires shuffling, (ii) it needs data transmission to broadcast a single feature in each search step, and (iii) the fact that, by default, the dataset is divided into a number of partitions equal to the number of features m in the dataset may not be optimal for all cases (while this parameter can be tuned, it can never exceed m). The main advantage of vertical partitioning is that the data layout and the broadcasting of the compared feature move all the information needed to calculate the contingency table to the same node, which means that this information can be more efficiently processed locally. Another advantage is that the whole dataset does not need to be read every time a new set of features has to be compared, since the dataset can be filtered by rows to process only the required features.

Due to the nature of the search strategy (best-first) used in the CFS, the first search step will always involve all features, so no filtering can be performed. For each subsequent step, only one more feature per step can be filtered out. This is especially important with high dimensionality datasets: the fact that the number of features is much higher than the number of search steps means that the percentage of features that can be filtered out is reduced.

A number of experiments were performed to quantify the effects of the advantages and disadvantages of each approach and to check the conditions in which one approach was better than the other.

7.2 Experiments

The experiments tested and compared time-efficiency and scalability for the horizontal and vertical DiCFS approaches so as to check whether they improved on the original non-distributed version of the CFS. In addition, tests and comparisons of execution times with those reported in the research made by [47] were also performed.

Analogously with DiReliefF, no experiments were needed to compare the quality of the results for the distributed and non-distributed CFS versions as the distributed versions were designed to return the same results as the original algorithm.

The experiments were performed on a single master node and up to ten slave nodes from the

big data platform of the Galician Supercomputing Technological Centre (CESGA). [2] The nodes have the following configuration:

- CPU: 2 X Intel Xeon E5-2620 v3 @ 2.40GHz
- CPU Cores: 12 (2X6)
- Total Memory: 64 GB
- Network: 10GbE
- Master Node Disks: 8 X 480GB SSD SATA 2.5" MLC G3HS
- Slave Node Disks: 12 X 2TB NL SATA 6Gbps 3.5" G2HS
- Java version: OpenJDK 1.8
- Spark version: 1.6
- Hadoop (HDFS) version: 2.7.1
- WEKA version: 3.8.1

Regarding the datasets, the same group of datasets described in Table 6.1 in Chapter 6 were used. With respect to algorithm parameter configuration, two defaults were used in all the experiments: the inclusion of locally predictive features and the use of five consecutive fails as a stopping criterion. These defaults apply to both distributed and non-distributed versions. Moreover, for the vertical partitioning version, the number of partitions was equal to the number of features, as set by default in [132]. The horizontally and vertically distributed versions of the CFS are labeled *DiCFS-hp* and *DiCFS-vp*, respectively.

First, the execution times for the four algorithms in the datasets using ten slave nodes with all their cores available were compared. For the case of the non-distributed version of the CFS, the implementation provided in the WEKA platform [69] was used. The results are shown in Figure 7.3.

Note that, with the aim of offering a comprehensive view of execution time behavior, Figure 7.3 shows results for sizes larger than the 100% of the datasets. To achieve these sizes, the instances in each dataset were duplicated as many times as necessary. Its important to note that since ECBDL14 is a very large dataset, its temporal scale is different from that of the other datasets.

Regarding the non-distributed version of the CFS, Figure 7.3 does not show results for WEKA in the experiments on the ECBDL14 dataset, because it was impossible to execute that version in the CESGA platform due to memory requirements exceeding the available limits. This also occurred with the larger samples from the EPSILON dataset for both algorithms: DiCFS-vp and

[2] `http://bigdata.cesga.es/`

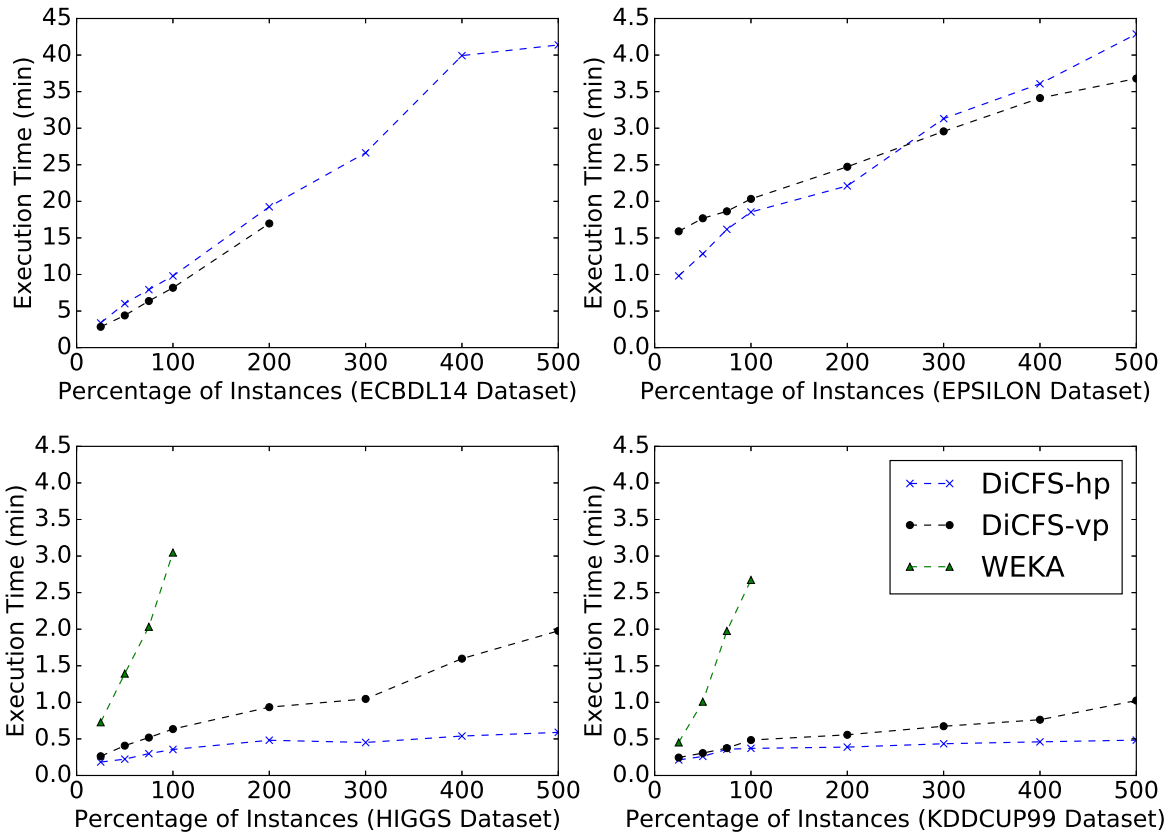

Figure 7.3: Execution time with respect to percentages of instances in four datasets, for DiCFS-hp and DiCFS-vp using ten nodes and for a non-distributed implementation in WEKA using a single node

DiCFS-hp. Even when it was possible to execute the WEKA version with the two smallest samples from the EPSILON dataset, these samples are not shown because the execution times were too high (19 and 69 minutes, respectively). Figure 7.3 shows successful results for the smaller HIGGS and KDDCUP99 datasets, which could still be processed in a single node of the cluster, as required by the non-distributed version. However, even in the case of these smaller datasets, the execution times of the WEKA version were worse compared to those of the distributed versions.

Regarding the distributed versions, DiCFS-vp was unable to process the oversized versions of the ECBDL14 dataset, due to the large amounts of memory required to perform shuffling. The HIGGS and KDDCUP99 datasets showed an increasing difference in favor of DiCFS-hp, this was due to the fact that these datasets have much smaller feature sizes than ECBDL14 and EPSILON. As mentioned earlier, DiCFS-vp ties parallelization to the number of features in the dataset, so datasets with small numbers of features were not able to fully leverage the cluster nodes. Another view of the same issue is given by the results for the EPSILON dataset; in this case, DiCFS-vp obtained the best execution times for the 300% sized and larger datasets. This was because there were too many partitions (2,000) for the number of instances available

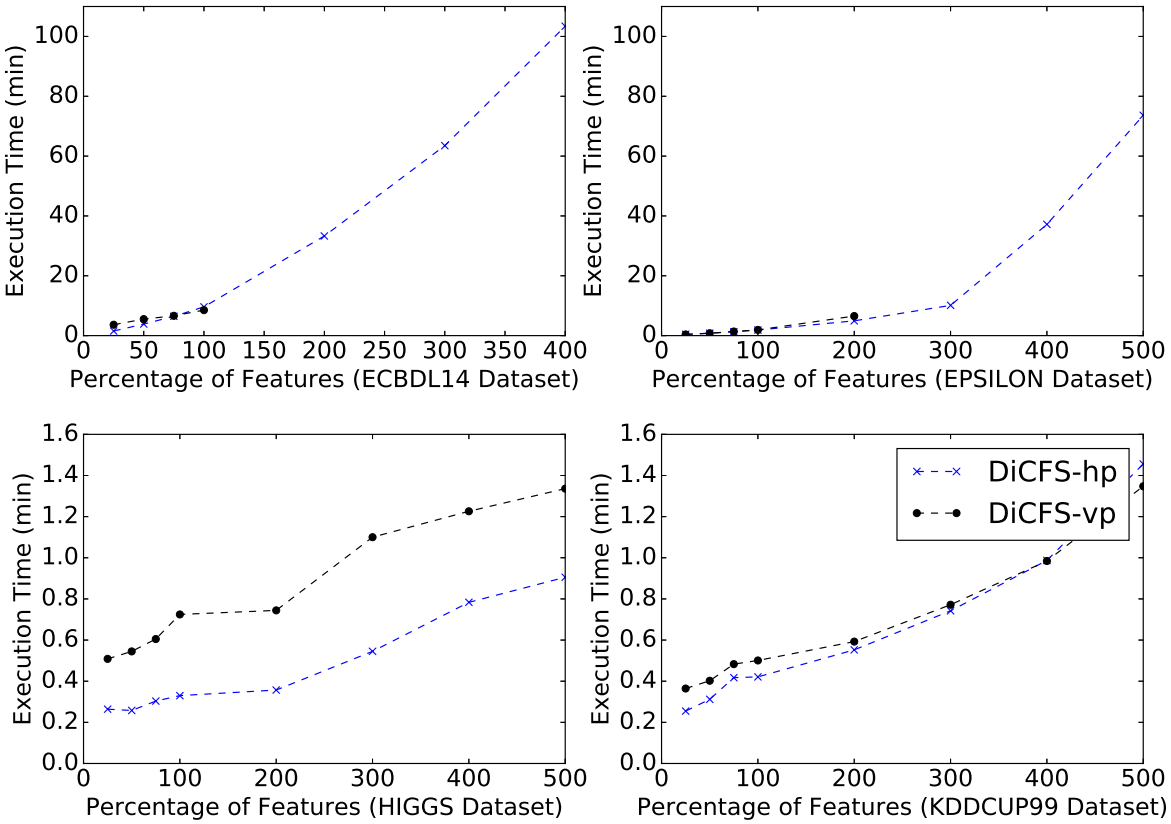

Figure 7.4: Execution times with respect to different percentages of features in four datasets for DiCFS-hp and DiCFS-vp

in smaller than 300% sized datasets; further experiments showed that adjusting the number of partitions to 100 reduced the execution time of DiCFS-vp for the 100% EPSILON dataset from about 2 minutes to 1.4 minutes (faster than DiCFS-hp). Reducing the number of partitions further, however, caused the execution time to start increasing again.

Figure 7.4 shows the results for similar experiments, except that this time the percentage of features in the datasets was varied and the features were copied to obtain oversized versions of the datasets. It can be observed that the number of features had a greater impact on the memory requirements of DiCFS-vp. This caused problems not only in processing the ECBDL14 dataset but also the EPSILON dataset. A quadratic time complexity in the number of features can be observed and how the temporal scale in the EPSILON dataset (with the highest number of dimensions) matches that of the ECBDL14 dataset. As for the KDDCUP99 dataset, the results show that increasing the number of features obtained a better level of parallelization and a slightly improved execution time of DiCFS-vp compared to DiCFS-hp for the 400% dataset version and above.

An important measure of the scalability of an algorithm is *speed-up*, which is a measure that

indicates how capable an algorithm is of leveraging a growing number of nodes so as to reduce execution times. The speed-up definition used is shown in Equation (7.2), all the available cores for each node (i.e., 12) were used. The experimental results are shown in Figure 7.5, where it can be observed that, for all four datasets, DiCFS-hp scales better than DiCFS-vp. It can also be observed that the HIGGS and KDDCUP datasets are too small to take advantage of the use of more than two nodes and also that practically no speed-up improvement is obtained from increasing this value.

To summarize, the experiments show that even when vertical partitioning results in shorter execution times (the case in certain circumstances, e.g., when the dataset has an adequate number of features and instances for optimal parallelization according to the cluster resources), the benefits are not significant and may even be eclipsed by the effort invested in determining whether this approach is indeed the most efficient approach for a particular dataset or a particular hardware configuration or in fine-tuning the number of partitions. Horizontal partitioning should therefore be considered as the best option in the general case.

$$speedup(m) = \left\lfloor \frac{execution\ \ time\ \ on\ \ 2\ \ nodes}{execution\ \ time\ \ on\ \ m\ \ nodes} \right\rfloor \tag{7.2}$$

Lastly, a comparison of the DiCFS-hp approach with that of Eiras-Franco et al. [47], was performed. In [47] the authors describe a Spark-based distributed version of the CFS for regression problems. The comparison was based on their experiments with the HIGGS and EPSILON datasets but using hardware available. Those datasets were selected as only having numerical features and so could naturally be treated as regression problems. Table 7.1 shows execution time and speed-up values obtained for different sizes of both datasets for both distributed and non-distributed versions and considering them to be classification and regression problems. Regression-oriented versions for the Spark and WEKA versions are labeled RegCFS and RegWEKA, respectively, the number after the dataset name represents the sample size and the letter indicates whether the sample had removed or added instances (i) or removed or added features (f). In the case of oversized samples, the method used was the same as described above, i.e., features or instances were copied as necessary. The experiments were performed using ten cluster nodes for the distributed versions and a single node for the WEKA version. The resulting speed-up was calculated as the WEKA execution time divided by the corresponding Spark execution time.

The original experiments in [47] were performed only using EPSILON_50i and HIGGS_100i. It can be observed that much better speed-up was obtained by the DiCFS-hp version for EPSILON_50i but in the case of HIGGS_100i, the resulting speed-up in the classification version was lower than the regression version. However, in order to have a better comparison, two more versions for each dataset were considered, Table 7.1 shows that the DiCFS-hp version has a better speed-up in all cases except in HIGGS_100i dataset mentioned before.

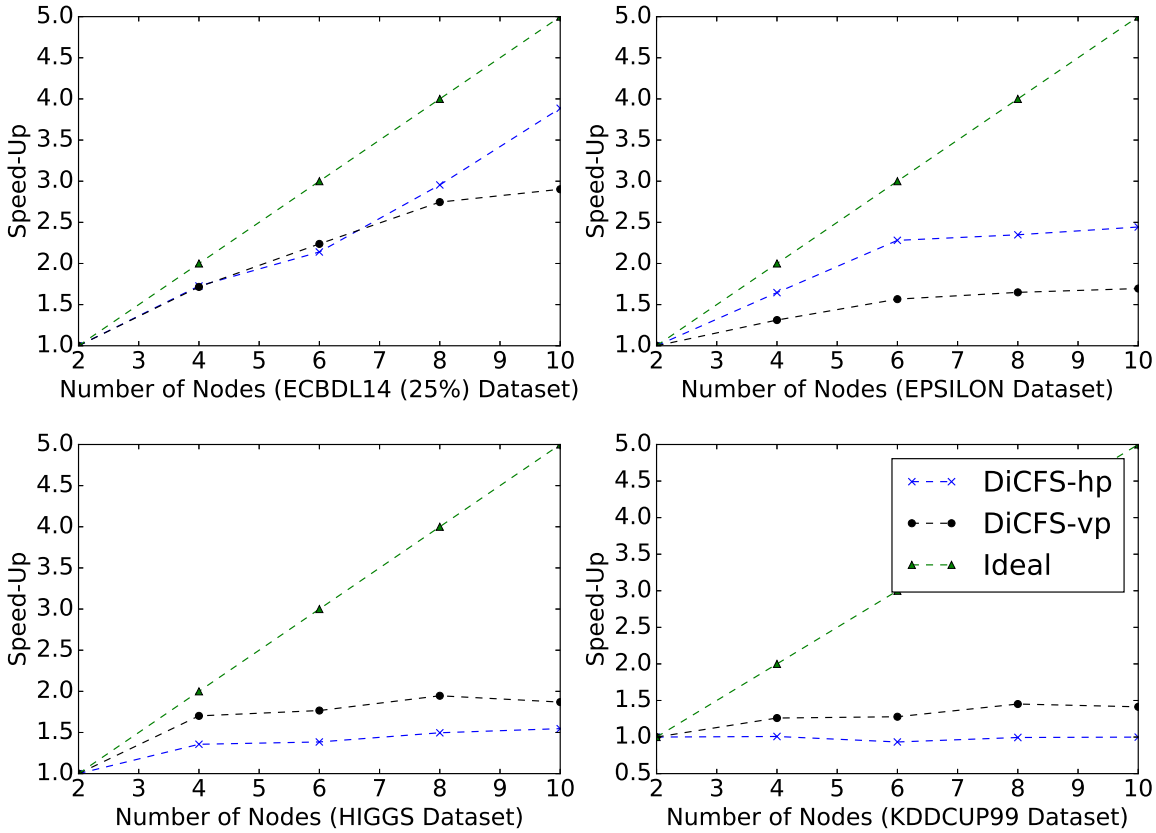

Figure 7.5: Speed-up for four datasets for DiCFS-hp and DiCFS-vp

Table 7.1: Execution time and speed-up values for different CFS versions for regression and classification

Dataset	Execution Time (sec)				Speed-Up	
	WEKA	RegWEKA	DiCFS-hp	RegCFS	RegCFS	DiCFS-hp
EPSILON_25i	1011.42	655.56	58.85	63.61	10.31	17.19
EPSILON_25f	393.91	703.95	25.83	55.08	12.78	15.25
EPSILON_50i	4103.35	2228.64	76.98	110.13	20.24	53.30
HIGGS_100i	182.86	327.61	21.34	23.70	13.82	8.57
HIGGS_200i	2079.58	475.98	28.89	26.77	17.78	71.99
HIGGS_200f	934.07	720.32	21.42	34.35	20.97	43.61

Part III

Conclusions and Future Work

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This section describes the conclusions drawn by the author after following the research process required in the elaboration of this dissertation. This process started with the study and analysis of the theoretical concepts about data mining, machine learning, big data and feature selection, continued with the definition of its objectives and finished with the design, test and publication of the contributions described in the previous part. Also, scientific work can never be considered as ended, there is always space for improvement and there are always promising paths to be followed. This future work is also stated in this chapter.

The chapter is divided in three small sections, the first two respectively present the specific conclusions and future work for each of the two contributions presented in the previous part. Finally, the last section presents conclusions and future work from a general point of view.

The referred contributions were described and cited in Section 1.2.

8.1 DiReliefF: Conclusions and Future Work

Chapter 6 of this thesis presented DiRelieF, a distributed version of the well-known ReliefF feature selection algorithm. This version was implemented using the Apache Spark programming model to deal with current Big Data requirements such as failure recovery and scalability over a cluster of commodity computers. Even when the ReliefF algorithm is easily parallelizable by associating jobs to each independent iteration and then merging its results, this method ties the number of jobs to the sample size (number of iterations) and requires an equal number of passes through the dataset. For this reason an alternative version based on the Spark *map* and *aggregate* operations and on the use of binary heaps was designed. This version does not suffer the problems mentioned and requires a single pass through the whole dataset to perform the

main operations of the algorithm compared to the m passes required by the original version.

As part of the experimental work, the proposal was compared with a non distributed version of the algorithm implemented on the WEKA platform. The results showed that the non distributed version is poorly scalable, i.e., it is unable to handle large datasets due to memory requirements. Conversely, DiReliefF is fully scalable and provides better execution times and memory usage when dealing with large datasets. Experiments also showed that the algorithm is capable of returning stable results with sample sizes that are much smaller than the size of the complete dataset.

Regarding future work, two promising lines are the following:

- Define a statistically sound method for selecting the sample size m, also even when it was shown that the method becomes stable with small sample sizes, it would be very important to determine how much the sample size in large datasets affects the quality of the results.
- The most resource demanding step in the algorithm is the calculation of the nearest neighbors of the selected sample instance in all classes. An important future line of research is to evaluate the application of techniques that return approximate nearest neighbors such as Locally Sensitive Hashing, and determine how does its use affect the final results of the ReliefF algorithm.
- As mentioned in Section 5.2.1, many versions have been derived from the Relief algorithm some of them for tasks other than binary and multi-class classification (ReliefF), subsequent research work could be made in the adaptation of DiRelieF to work as a scalable version for these derived versions and possibly serve to other types of tasks such as multi-instance and multi-label learning.

8.2 DiCFS: Conclusions and Future Work

Chapter 7 described two parallel and distributed versions of the CFS filter-based feature selection algorithm using the Apache Spark programming model: DiCFS-hp and DiCFS-vp. These two versions essentially differ in how the dataset is distributed across the nodes of the cluster. The first version distributes the data by splitting rows (instances) and the second version, following [132], distributes the data by splitting columns (features). As the outcome of a four-way comparison of DiCFS-vp and DiCFS-hp, a non-distributed implementation in WEKA and a distributed regression version in Spark, the following conclusions can be drawn:

- As was expected, both DiCFS-vp and DiCFS-hp were able to handle larger datasets in much a more time-efficient manner than the classical WEKA implementation. Moreover, in many cases they were the only feasible way to process certain types of datasets because of prohibitive WEKA memory requirements.

- Of the horizontal and vertical partitioning schemes, the horizontal version (DiCFS-hp) proved to be the best option in the general case due to its greater scalability and its natural partitioning mode that enables the Spark framework to make better use of cluster resources.
- For classification problems, the benefits obtained from distribution compared to non-distribution version can be considered equal to or even better than the benefits already demonstrated for the regression domain [47].

Regarding future research, an especially interesting line is whether it is necessary for this kind of algorithm to process all the data available or whether it would be possible to design automatic sampling procedures that could guarantee that, under certain circumstances, equivalent results could be obtained. In the case of the CFS, this question becomes more pertinent in view of the study of symmetrical uncertainty in datasets with up to 20,000 samples by Hall [70], where tests showed that symmetrical uncertainty decreased exponentially with the number of instances and then stabilized at a certain number. Another line of future work could be research into different data partitioning schemes that could, for instance, improve the locality of data while overcoming the disadvantages of vertical partitioning.

8.3 General Conclusions and Future Work

Going back to Section 1.1 were the research objective was formulated, and observing the results obtained from both designed algorithms, it is possible to conclude that the objective of "developing new versions of existing important feature selection methods so that they are able to cope with large datasets in a scalable fashion" was successfully accomplished for the specific cases of ReliefF and CFS feature selection algorithms. Both versions are ready to serve as valuable tools to other researchers and practitioners in different fields that may want to process large datasets for their own objectives.

Respecting the platform used for the implementation of the algorithms, i.e.: Apache Spark, it has been of great aid, the high level concepts the framework provides had made possible the implementation of the algorithms without worrying about the common lower level details such as thread synchronizations and failure recovery. Its not a surprise that by the time of writing this last section Apache Spark has consolidated as one of the most important large data processing tools available.

Finally, the approach followed in this work can also be tested with other widely used feature selection algorithms. It is clear that each method imposes its own challenges to over pass, but the current work can form part of a background for future works that may archive similar or better results with other algorithms.

Bibliography

[1] Apache Spark Web Site.
URL spark.apache.org.

[2] Small and Midsize Companies Look to Make Big Gains With Big Data, 2012.
URL http://global.sap.com/news-reader/index.epx?PressID=19188.

[3] MongoDB FAQ Page, 2018.
URL https://www.mongodb.com/faq{#}consistency.

[4] David W. Aha, Dennis Kibler, and Marc K. Albert.
Instance-Based Learning Algorithms.
Machine Learning, 6(1):37–66, 1991.
ISSN 15730565.
doi: 10.1023/A:1022689900470.

[5] Amir H Alavi and Amir H Gandomi.
Big data in civil engineering.
Automation in Construction, 79:1–2, 2017.
ISSN 0926-5805.
doi: https://doi.org/10.1016/j.autcon.2016.12.008.
URL http://www.sciencedirect.com/science/article/pii/S0926580516305246.

[6] Jaume Bacardit, Paweł Widera, Alfonso Márquez-chamorro, Federico Divina, Jesús S. Aguilar-Ruiz, and Natalio Krasnogor.
Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features.
Bioinformatics, 28(19):2441–2448, 2012.
ISSN 13674803.
doi: 10.1093/bioinformatics/bts472.

[7] Eman M. Bahgat, Sherine Rady, Walaa Gad, and Ibrahim F. Moawad.
Efficient email classification approach based on semantic methods.
Ain Shams Engineering Journal, 9(4):3259–3269, 2018.
ISSN 20904479.

doi: 10.1016/j.asej.2018.06.001.

[8] Madhushri Banerjee and Sumit Chakravarty.
Privacy Preserving Feature Selection for Distributed Data Using Virtual Dimension.
In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 2281–2284, New York, NY, USA, 2011. ACM.
ISBN 978-1-4503-0717-8.
doi: 10.1145/2063576.2063946.
URL http://doi.acm.org/10.1145/2063576.2063946.

[9] Blaise Barney.
Introduction to parallel computing.
Lawrence Livermore National Laboratory, 6(13):10, 2010.

[10] R Bellman.
Dynamic Programming.
Rand Corporation research study. Princeton University Press, 1957.
ISBN 9780691079516.
URL https://books.google.it/books?id=wdtoPwAACAAJ.

[11] R Bellman.
Adaptive Control Processes: A Guided Tour.
Princeton Legacy Library. Princeton University Press, 1961.

[12] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos.
Distributed feature selection: An application to microarray data classification.
Applied Soft Computing, 30:136–150, may 2015.
ISSN 15684946.
doi: 10.1016/j.asoc.2015.01.035.
URL http://linkinghub.elsevier.com/retrieve/pii/S156849461500054X.

[13] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos.
Recent advances and emerging challenges of feature selection in the context of big data.
Knowledge-Based Systems, 86:33–45, 2015.
ISSN 09507051.
doi: 10.1016/j.knosys.2015.05.014.

[14] Verónica Bolón-Canedo.
Novel feature selection methods for high dimensional data.
PhD thesis, 2014.

[15] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos.

A review of feature selection methods on synthetic data.
Knowledge and Information Systems, 34(3):483–519, 2012.
ISSN 0219-3116.
doi: 10.1007/s10115-012-0487-8.
URL http://dx.doi.org/10.1007/s10115-012-0487-8.

[16] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Joana Cerviño-Rabuñal.
Scaling Up Feature Selection: A Distributed Filter Approach.
pages 121–130. Springer Berlin Heidelberg, 2013.
doi: 10.1007/978-3-642-40643-0_13.
URL http://link.springer.com/10.1007/978-3-642-40643-0{_}13.

[17] André B. Bondi.
Characteristics of scalability and their impact on performance.
In *Proceedings of the second international workshop on Software and performance - WOSP '00*, pages 195–203, New York, New York, USA, 2000. ACM Press.
ISBN 158113195X.
doi: 10.1145/350391.350432.
URL http://portal.acm.org/citation.cfm?doid=350391.350432.

[18] Noorhannah Boodhun and Manoj Jayabalan.
Risk prediction in life insurance industry using supervised learning algorithms.
Complex & Intelligent Systems, 4(2):145–154, jun 2018.
ISSN 2199-4536.
doi: 10.1007/s40747-018-0072-1.

[19] Dhruba Borthakur and Others.
HDFS architecture guide.
Hadoop Apache Project, 53:1–13, 2008.

[20] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst.
HaLoop.
Proceedings of the VLDB Endowment, 3(1-2):285–296, sep 2010.
ISSN 21508097.
doi: 10.14778/1920841.1920881.
URL http://dl.acm.org/citation.cfm?doid=1920841.1920881.

[21] Jingjing Cao, Kai Liu, Lin Liu, Yuanhui Zhu, Jun Li, and Zhi He.
Identifying mangrove species using field close-range snapshot hyperspectral imaging and machine-learning techniques.
Remote Sensing, 10(12), 2018.

ISSN 20724292.
doi: 10.3390/rs10122047.

[22] Rich Caruana and Alexandru Niculescu-Mizil.
An empirical comparison of supervised learning algorithms.
Proceedings of the 23rd international conference on Machine learning, C(1):161–168, 2006.
ISSN 1595933832.
doi: 10.1145/1143844.1143865.
URL http://portal.acm.org/citation.cfm?doid=1143844.1143865.

[23] John M. Chambers.
Greater or lesser statistics: a choice for future research.
Statistics and Computing, 3(4):182–184, 1993.
ISSN 09603174.
doi: 10.1007/BF00141776.

[24] Varun Chandola, Arindam Banerjee, and Vipin Kumar.
Anomaly detection: A survey.
ACM computing surveys (CSUR), 41(3):15, 2009.

[25] Girish Chandrashekar and Ferat Sahin.
A survey on feature selection methods.
Computers & Electrical Engineering, 40(1):16–28, jan 2014.
ISSN 00457906.
doi: 10.1016/j.compeleceng.2013.11.024.
URL http://www.sciencedirect.com/science/article/pii/S0045790613003066.

[26] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien.
Semi-Supervised Learning.
The MIT Press, 1st edition, 2006.
ISBN 0262514125, 9780262514125,.

[27] Cisco/Systems.
The Zettabyte Era — Trends and Analysis, 2016.
URL http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.

[28] William S. Cleveland.
Data science: An action plan for expanding the technical areas of the field of statistics.
Statistical Analysis and Data Mining, 7(6):414–417, 2014.
ISSN 19321872.
doi: 10.1002/sam.11239.

[29] Gordon V. Cormack.
Email Spam Filtering: A Systematic Review.
Foundations and Trends® in Information Retrieval, 1(4):335–455, 2008.
ISSN 1554-0669.
doi: 10.1561/1500000006.
URL http://www.nowpublishers.com/article/Details/INR-006.

[30] George Coulouris, Jean Dollimore, and Tim Kindberg.
Distributed Systems: Concepts and Design.
Pearson, 5 edition, 2011.
ISBN 978-0132143011.

[31] Michael Cox and David Ellsworth.
Application-Controlled Demand Paging for Out-of-Core Visualization.
Proceedings of the 8th IEEE Visualization '97 Conference, pages 235–244, 1997.

[32] Kamalika Das, Kanishka Bhaduri, and Hillol Kargupta.
A local asynchronous distributed privacy preserving feature selection algorithm for large peer-to-peer networks.
Knowledge and Information Systems, 24(3):341–367, sep 2010.
ISSN 0219-3116.
doi: 10.1007/s10115-009-0274-3.
URL https://doi.org/10.1007/s10115-009-0274-3.

[33] Manoranjan Dash and Huan Liu.
Consistency-based search in feature selection.
Artificial Intelligence, 151(1-2):155–176, dec 2003.
ISSN 00043702.
doi: 10.1016/S0004-3702(03)00079-1.
URL http://linkinghub.elsevier.com/retrieve/pii/S0004370203000791.

[34] Sujata Dash and B N Patra.
Redundant gene selection based on genetic and quick-reduct algorithms.
International Journal on Data Mining and Intelligent Information Technology Applications, 3(2):1–9, 2013.

[35] Thomas H Davenport.
Competing on analytics.
Harvard Business Review, 84(1):98, 2006.

[36] Thomas H Davenport.
Analytics 3.0.

Harvard Business Review, 91(12):64———+, 2013.

[37] L Nunes De Castro and Fernando J Von Zuben.
The clonal selection algorithm with engineering applications.
In *Proceedings of GECCO*, volume 2000, pages 36–39, 2000.

[38] Jeffrey Dean and Sanjay Ghemawat.
MapReduce: Simplified Data Processing on Large Clusters.
Communications of the ACM, 51(1):107, 2008.
URL http://dl.acm.org/citation.cfm?id=1327452.1327492.

[39] Joaquín Derrac, Salvador García, and Francisco Herrera.
A Survey on Evolutionary Instance Selection and Generation.
International Journal of Applied Metaheuristic Computing, 1(1):60–92, 2010.
ISSN 1947-8283.
doi: 10.4018/jamc.2010102604.
URL http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jamc.2010102604.

[40] Vasant Dhar.
Data science and prediction.
Communications of the ACM, 56(12):64–73, 2013.
ISSN 00010782.
doi: 10.1145/2500499.
URL http://dl.acm.org/citation.cfm?doid=2534706.2500499.

[41] Francis X. Diebold.
A Personal Perspective on the Origin(s) and Development of 'Big Data': The Phenomenon, the Term, and the Discipline, Second Version.
SSRN Electronic Journal, 2012.
ISSN 1556-5068.
doi: 10.2139/ssrn.2202843.
URL http://www.ssrn.com/abstract=2202843.

[42] Chelsea Dobbins and Reza Rawassizadeh.
Towards Clustering of Mobile and Smartwatch Accelerometer Data for Physical Activity Recognition.
Informatics, 5(2):29, jun 2018.
ISSN 2227-9709.
doi: 10.3390/informatics5020029.

[43] David Donoho.

50 Years of Data Science.
R Software, page 41, 2015.
ISSN 1061-8600.
doi: 10.1080/10618600.2017.1384734.

[44] Bruce Draper, Carol Kaito, and Jose Bins.
Iterative Relief.
In *2003 Conference on Computer Vision and Pattern Recognition Workshop*, pages 62–62. IEEE, jun 2003.
doi: 10.1109/CVPRW.2003.10065.
URL http://ieeexplore.ieee.org/document/4624323/.

[45] R O Duda, P E Hart, and D G Stork.
Pattern Classification.
John Wiley & Sons, 2001.
ISBN 0471056693.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.1318{&}rep=rep1{&}type=pdf.

[46] Kalpana Dwivedi and Sanjay Kumar Dubey.
Analytical review on Hadoop Distributed file system.
Proceedings of the 5th International Conference on Confluence 2014: The Next Generation Information Technology Summit, pages 174–181, 2014.
ISSN 13869477.
doi: 10.1109/CONFLUENCE.2014.6949336.

[47] Carlos Eiras-Franco, Verónica Bolón-Canedo, Sabela Ramos, Jorge González-Domínguez, Amparo Alonso-Betanzos, and Juan Touriño.
Multithreaded and Spark parallelization of feature selection filters.
Journal of Computational Science, 17:609–619, nov 2016.
ISSN 18777503.
doi: 10.1016/j.jocs.2016.07.002.
URL http://linkinghub.elsevier.com/retrieve/pii/S1877750316301107.

[48] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-hee Bae, Judy Qiu, and Geoffrey Fox.
Twister : A Runtime for Iterative MapReduce.
In *Proceedings of the ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.
ISBN 9781605589428.
doi: 10.1145/1851476.1851593.

[49] Margaret J. Eppstein and Paul Haake.
Very large scale ReliefF for genome-wide association analysis.
In *2008 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 112–119. IEEE, sep 2008.
ISBN 978-1-4244-1778-0.
doi: 10.1109/CIBCB.2008.4675767.
URL http://ieeexplore.ieee.org/document/4675767/.

[50] Ahmed K. Farahat, Ahmed Elgohary, Ali Ghodsi, and Mohamed S. Kamel.
Distributed Column Subset Selection on MapReduce.
In *2013 IEEE 13th International Conference on Data Mining*, pages 171–180. IEEE, dec 2013.
ISBN 978-0-7695-5108-1.
doi: 10.1109/ICDM.2013.155.
URL http://ieeexplore.ieee.org/document/6729501/.

[51] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth.
From Data Mining to Knowledge Discovery in Databases.
AI Magazine, 17(3):37, 1996.
ISSN 0738-4602.

[52] Usama M Fayyad and Keki B Irani.
Multi-Interval Discretization of Continuos-Valued Attributes for Classification Learning, 1993.
ISSN 15730565.
URL http://trs-new.jpl.nasa.gov/dspace/handle/2014/35171.

[53] Amir Gandomi and Murtaza Haider.
Beyond the hype: Big data concepts, methods, and analytics.
International Journal of Information Management, 35(2):137–144, apr 2015.
ISSN 0268-4012.
doi: 10.1016/J.IJINFOMGT.2014.10.007.
URL https://www.sciencedirect.com/science/article/pii/S0268401214001066{#}bbib0110.

[54] John Gantz and David Reinsel.
Extracting Value from Chaos State of the Universe: An Executive Summary.
IDC iView, (June):1–12, 2011.
URL http://idcdocserv.com/1142.

[55] Daniel J Garcia, Lawrence O Hall, Dmitry B Goldgof, and Kurt Kramer.

A Parallel Feature Selection Algorithm from Random Subsets.
In *Proceedings of the intl. workshop on parallel data mining*, pages 1–12, 2004.

[56] S García, J Luengo, J A Sáez, V López, and F Herrera.
A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning.
IEEE Transactions on Knowledge and Data Engineering, 25(4):734–750, apr 2013.
ISSN 1041-4347.
doi: 10.1109/TKDE.2012.35.

[57] Salvador García, Julián Luengo, and Francisco Herrera.
Feature Selection.
In *Data Preprocessing in Data Mining*, pages 163–193. Springer International Publishing, Cham, 2015.
ISBN 978-3-319-10247-4.
doi: 10.1007/978-3-319-10247-4_7.
URL http://dx.doi.org/10.1007/978-3-319-10247-4{_}7.

[58] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.
The Google file system.
In *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, 2003.
ISBN 1581137575.
doi: 10.1145/945449.945450.

[59] E E Ghiselli.
Theory of Psychological Measurement.
McGraw-Hill series in psychology. McGraw-Hill, 1964.
ISBN 9780070231405.
URL https://books.google.es/books?id=mmh9AAAAMAAJ.

[60] Jorge González-Domínguez, Verónica Bolón-Canedo, Borja Freire, and Juan Touriño.
Parallel feature selection for distributed-memory clusters.
Information Sciences, jan 2019.
ISSN 0020-0255.
doi: 10.1016/J.INS.2019.01.050.
URL https://www.sciencedirect.com/science/article/pii/S0020025519300635.

[61] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning.
MIT Press, 2016.
URL http://www.deeplearningbook.org/.

[62] Delaney Granizo-Mackenzie and Jason H Moore.
Multiple Threshold Spatially Uniform ReliefF for the Genetic Analysis of Complex Human Diseases.
In Leonardo Vanneschi, William S Bush, and Mario Giacobini, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 1–10, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
ISBN 978-3-642-37189-9.

[63] Casey S Greene, Nadia M Penrod, Jeff Kiralis, and Jason H Moore.
Spatially Uniform ReliefF (SURF) for computationally-efficient filtering of gene-gene interactions.
BioData Mining, 2(1):5, 2009.
ISSN 1756-0381.
doi: 10.1186/1756-0381-2-5.
URL http://biodatamining.biomedcentral.com/articles/10.1186/1756-0381-2-5.

[64] Casey S Greene, Daniel S Himmelstein, Jeff Kiralis, and Jason H Moore.
The Informative Extremes: Using Both Nearest and Farthest Individuals Can Improve Relief Algorithms in the Domain of Human Genetics.
In Clara Pizzuti, Marylyn D Ritchie, and Mario Giacobini, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 182–193, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
ISBN 978-3-642-12211-8.

[65] Quanquan Gu, Zhenhui Li, and Jiawei Han.
Generalized fisher score for feature selection.
arXiv preprint arXiv:1202.3725, 2012.

[66] Isabelle Guyon and André Elisseeff.
An Introduction to Variable and Feature Selection.
Journal of Machine Learning Research (JMLR), 3(3):1157–1182, 2003.
ISSN 00032670.
doi: 10.1016/j.aca.2011.07.027.

[67] Peter Haig and Dietmar Krick.
Data at Edge. Watson down to Earth, 2015.
URL https://www-935.ibm.com/services/multimedia/Vortrag{_}IBM{_}Peter-Krick.pdf.

[68] Alon Halevy, Peter Norvig, and Fernando Pereira.

Unreasonable Effectiveness of Data.
2009.
ISSN 15411672.
doi: 10.1109/MIS.2009.36.

[69] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian Witten.
The WEKA data mining software: An update.
SIGKDD Explorations, 11(1):10–18, 2009.
ISSN 19310145.
doi: 10.1145/1656274.1656278.

[70] Mark A. Hall.
Correlation-based feature selection for machine learning.
PhD Thesis., Department of Computer Science, Waikato University, New Zealand, (April), 1999.
doi: 10.1.1.37.4643.

[71] Mark A. Hall.
Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning.
pages 359–366, jun 2000.
URL `http://dl.acm.org/citation.cfm?id=645529.657793`.

[72] Kuk-Hyun Han and Jong-Hwan Kim.
Quantum-inspired evolutionary algorithms with a new termination criterion, H/sub/spl epsi//gate, and two-phase scheme.
IEEE transactions on evolutionary computation, 8(2):156–169, 2004.

[73] Harlan Harris, Sean Murphy, and Marck Vaisman.
Analyzing the Analyzers: An Introspective Survey of Data Scientists and Their Work.
" O'Reilly Media, Inc.", 2013.

[74] Jerónimo Hernández-González, Iñaki Inza, and Jose A. Lozano.
Weak supervision and other non-standard classification problems: A taxonomy.
Pattern Recognition Letters, 69:49–55, 2016.
ISSN 01678655.
doi: 10.1016/j.patrec.2015.10.008.

[75] Tin Kam Ho.
Random Decision Forests.

In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278———, Washington, DC, USA, 1995. IEEE Computer Society.
ISBN 0-8186-7128-9.
URL http://dl.acm.org/citation.cfm?id=844379.844681.

[76] Victoria J. Hodge, Simon O'Keefe, and Jim Austin.
Hadoop neural network for parallel and distributed feature selection.
Neural Networks, 78:24–35, 2016.
ISSN 18792782.
doi: 10.1016/j.neunet.2015.08.011.
URL http://dx.doi.org/10.1016/j.neunet.2015.08.011.

[77] Se June Hong.
Use of Contextual Information for Feature Ranking and Discretization.
IEEE Trans. on Knowl. and Data Eng., 9(5):718–730, sep 1997.
ISSN 1041-4347.
doi: 10.1109/69.634751.
URL http://dx.doi.org/10.1109/69.634751.

[78] IDC / EMC.
The Digital Universe of Opportunities.
IDC / EMC Report, page 17, 2014.
URL http://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf.

[79] Ephraim T. Iorkyase, Christos Tachtatzis, Ian A. Glover, and Robert C. Atkinson.
RF-based location of partial discharge sources using received signal features.
High Voltage, 4(1):28–32, mar 2018.
ISSN 2397-7264.
doi: 10.1049/hve.2018.5027.

[80] Manar Jaradat, Moath Jarrah, Abdelkader Bousselham, Yaser Jararweh, and Mahmoud Al-Ayyoub.
The Internet of Energy: Smart Sensor Networks and Big Data Management for Smart Grid.
Procedia Computer Science, 56:592–597, 2015.
ISSN 1877-0509.
doi: https://doi.org/10.1016/j.procs.2015.07.250.
URL http://www.sciencedirect.com/science/article/pii/S1877050915017317.

[81] Licheng Jiao, Yangyang Li, Maoguo Gong, and Xiangrong Zhang.
Quantum-inspired immune clonal algorithm for global optimization.
IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 38(5):1234–1253, 2008.

[82] Enric Junqué de Fortuny, David Martens, and Foster Provost.
Predictive Modeling with Big Data.
Mary Ann Liebert Inc, 1(4), 2013.
doi: 10.1089/big.2013.0037.
URL https://www.liebertpub.com/doi/pdf/10.1089/big.2013.0037.

[83] Alexandros Kalousis, Julien Prados, and Melanie Hilario.
Stability of feature selection algorithms: a study on high-dimensional spaces.
Knowledge and Information Systems, 12(1):95–116, 2006.
ISSN 0219-3116.
doi: 10.1007/s10115-006-0040-8.
URL http://dx.doi.org/10.1007/s10115-006-0040-8.

[84] James; Kennedy and Russell; Eberhart.
Particle swarm optimization.
Encyclopedia of Machine Learning, pages 1942–1948, 1995.
ISSN 1935-3812.
doi: 10.1007/s11721-007-0002-0.

[85] Gang-Hoon Kim, Silvana Trimi, and Ji-Hyong Chung.
Big-data applications in the government sector.
Communications of the ACM, 57(3):78–85, 2014.

[86] Kenji Kira and Larry A. Rendell.
A Practical Approach to Feature Selection.
Machine Learning Proceedings 1992, pages 249–256, jan 1992.
doi: 10.1016/B978-1-55860-247-2.50037-1.
URL https://www.sciencedirect.com/science/article/pii/B9781558602472500371.

[87] Kenji Kira and Larry A Rendell.
The feature selection problem: Traditional methods and a new algorithm.
In *Aaai*, volume 2, pages 129–134, 1992.

[88] Igor Kononenko.
Estimating attributes: Analysis and extensions of RELIEF.
Machine Learning: ECML-94, 784:171–182, 1994.

ISSN 03029743.
doi: 10.1007/3-540-57868-4.
URL http://www.springerlink.com/index/10.1007/3-540-57868-4.

[89] Jeremy Kubica, Sameer Singh, and Daria Sorokina.
Parallel Large-Scale Feature Selection.
In *Scaling Up Machine Learning*, number February, pages 352–370. 2011.
ISBN 9781139042918.
doi: 10.1017/CBO9781139042918.018.
URL http://ebooks.cambridge.org/ref/id/CBO9781139042918A143.

[90] L I Kuncheva.
A stability index for feature selection.
International Multi-conference: artificial intelligence and applications, pages 390–395, 2007.

[91] Doug Laney.
META Delta.
Application Delivery Strategies, 949(February 2001):4, 2001.
ISSN 09505849.
doi: 10.1016/j.infsof.2008.09.005.

[92] Pedro Larrañaga and Jose A Lozano.
Estimation of distribution algorithms: A new tool for evolutionary computation, volume 2.
Springer Science & Business Media, 2001.

[93] Chen Hang (Udacity) Lee.
3 Data Careers Decoded and What It Means for You, 2014.
URL https://blog.udacity.com/2014/12/data-analyst-vs-data-scientist-vs-data-engineer
html.

[94] Kwan-Yeung Lee, Pengfei Liu, Kwong-Sak Leung, and Man-Hon Wong.
Very large scale relieff algorithm on gpu for genome-wide association study.
In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 78. The Steering Committee of The World Congress in Computer Science, Computer˜. . . , 2015.

[95] David D Lewis.
Feature selection and feature extraction for text categorization.
In *Proceedings of the workshop on Speech and Natural Language*, pages 212–217. Association for Computational Linguistics, 1992.

[96] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu.
Feature Selection: A Data Perspective.
jan 2016.
URL http://arxiv.org/abs/1601.07996.

[97] K C Li, H Jiang, and A Y Zomaya.
Big Data Management and Processing.
Chapman & Hall/CRC Big Data Series. CRC Press, 2017.
ISBN 9781351650045.
URL https://books.google.hn/books?id=JVkkDwAAQBAJ.

[98] M Lichman.
UCI Machine Learning Repository, 2013.
URL http://archive.ics.uci.edu/ml.

[99] Huan Liu and Lei Yu.
Toward integrating feature selection algorithms for classification and clustering.
IEEE Transactions on Knowledge and Data Engineering, 17(4):491–502, apr 2005.
ISSN 1041-4347.
doi: 10.1109/TKDE.2005.66.

[100] Yang Liu, Lixiong Xu, and Maozhen Li.
The Parallelization of Back Propagation Neural Network in MapReduce and Spark.
International Journal of Parallel Programming, pages 1–20, feb 2016.
ISSN 0885-7458.
doi: 10.1007/s10766-016-0401-1.
URL http://link.springer.com/10.1007/s10766-016-0401-1.

[101] Mathew Lodge.
Cloudera and Hortonworks merger means Hadoop's influence is declining.
Venturebeat, 2018.
URL https://venturebeat.com/2018/10/06/cloudera-and-hortonworks-merger-means-hadoops-infl

[102] John Loughrey and Pádraig Cunningham.
Overfitting in Wrapper-Based Feature Subset Selection: The Harder You Try the Worse it Gets.
In Max Bramer, Frans Coenen, and Tony Allen, editors, *Research and Development in Intelligent Systems XXI*, pages 33–43, London, 2005. Springer London.
ISBN 978-1-84628-102-0.

[103] Heng Luo, Pierre Luc Carrier, Aaron Courville, and Yoshua Bengio.

Texture Modeling with Convolutional Spike-and-Slab RBMs and Deep Extensions.
Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS), 31:415–423, 2012.
ISSN 15337928.
URL http://arxiv.org/abs/1211.5687.

[104] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker.
Identifying Suspicious URLs : An Application of Large-Scale Online Learning.
In *Proceedings of the International Conference on Machine Learning (ICML)*, Montreal, Quebec, 2009.

[105] Frederic Magoules, Jie Pan, Kiat-An Tan, and Abhinit Kumar.
Introduction to grid computing.
CRC Press, 2009.
ISBN 978-1420074062.

[106] Ruchika Malhotra and Anjali Sharma.
Analyzing Machine Learning Techniques for Fault Prediction Using Web Applications.
Journal of Information Processing Systems, 14(3):751–770, jun 2018.
ISSN 1976-913X.
doi: 10.3745/JIPS.04.0077.

[107] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H Byers.
Big data: The next frontier for innovation, competition, and productivity, may 2011.
URL http://www.mckinsey.com/Insights/MGI/Research/Technology{%}5C{_}and{%}5C{_}Innovation/Big{%}5C{_}data{%}5C{_}The{%}5C{_}next{%}5

[108] Vivien Marx.
Biology: The big challenges of big data, 2013.

[109] John Mashey.
Big Data... and the Next Wave of InfraStress.
In *1999 USENIX Annual Technical Conference*, Monterey, California, 1999.
URL http://static.usenix.org/event/usenix99/invited{_}talks/mashey.pdf.

[110] Andrew McAfee and Erik Brynjolfsson.
Big data: the management revolution.
Harvard business review, 90(10):60–66,68,128, 2012.
ISSN 0017-8012.
doi: 10.1007/s12599-013-0249-5.
URL http://www.ncbi.nlm.nih.gov/pubmed/23074865.

[111] Brett A McKinney, Bill C White, Diane E Grill, Peter W Li, Richard B Kennedy, Gregory A Poland, and Ann L Oberg.
ReliefSeq: A Gene-Wise Adaptive-K Nearest-Neighbor Feature Selection Tool for Finding Gene-Gene Interactions and Main Effects in mRNA-Seq Gene Expression Data.
PLOS ONE, 8(12):1–12, 2013.
doi: 10.1371/journal.pone.0081527.
URL https://doi.org/10.1371/journal.pone.0081527.

[112] P M Mell and T Grance.
The NIST definition of cloud computing.
Technical report, National Institute of Standards and Technology, Gaithersburg, MD, 2011.
URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[113] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar.
MLlib: Machine Learning in Apache Spark.
CoRR, 2015.
URL http://arxiv.org/abs/1505.06807.

[114] Maged Michael, Jose E Moreira, Doron Shiloach, and Robert W Wisniewski.
Scale-up x scale-out: A case study using nutch/lucene.
In *2007 IEEE International Parallel and Distributed Processing Symposium*, page 441. IEEE, 2007.

[115] Tom M Mitchell.
Machine Learning.
Number 1. 1997.
ISBN 0070428077.
doi: 10.1145/242224.242229.
URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20{&}path=ASIN/0070428077.

[116] Pabitra Mitra, C A Murthy, and Sankar K Pal.
Unsupervised Feature Selection Using Feature Similarity.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI, 24(3):301–312, 2002.
ISSN 01628828.
doi: 10.1109/34.990133.
URL http://dx.doi.org/10.1109/34.990133.

[117] Jason H Moore and Bill C White.
Tuning ReliefF for Genome-Wide Genetic Analysis.
In Elena Marchiori, Jason H Moore, and Jagath C Rajapakse, editors, *Evolutionary Computation,Machine Learning and Data Mining in Bioinformatics*, pages 166–175, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
ISBN 978-3-540-71783-6.

[118] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic.
Deep learning applications and challenges in big data analytics.
Journal of Big Data, 2(1):1–21, 2015.
ISSN 21961115.
doi: 10.1186/s40537-014-0007-7.

[119] B Clifford Neuman.
Scale in Distributed Systems.
Readings in Distributed Computing Systems, pages 463–489, 1994.

[120] Hai Thanh Nguyen, Katrin Franke, and Slobodan Petrovic.
Improving Effectiveness of Intrusion Detection by Correlation Feature Selection.
INTERNATIONAL JOURNAL OF MOBILE COMPUTING AND MULTIMEDIA COMMUNICATIONS, 3(1):21–34, 2011.
ISSN 1937-9412.
doi: 10.4018/jmcmc.2011010102.

[121] Frederick L Oswald and Dan J Putka.
Big data methods in the social sciences.
Current Opinion in Behavioral Sciences, 18:103–106, 2017.
ISSN 2352-1546.
doi: https://doi.org/10.1016/j.cobeha.2017.10.006.
URL http://www.sciencedirect.com/science/article/pii/S2352154617300359.

[122] Raul Jose Palma-Mendoza, Daniel Rodriguez, and Luis De-Marcos.
Distributed ReliefF-based feature selection in Spark.
Knowledge and Information Systems, 57(1):1–20, jan 2018.
ISSN 02193116.
doi: 10.1007/s10115-017-1145-y.
URL http://link.springer.com/10.1007/s10115-017-1145-y.

[123] D. J. Patil and Thomas H. Davenport.
Data Scientist the sexiest job of the 21st century.

Harvard Business Review, (October), 2012.
URL https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century.

[124] Srini Penchikala.
Peter Cnudde on How Yahoo Uses Hadoop, Deep Learning and Big Data Platform.
InfoQ, 2016.
URL https://www.infoq.com/articles/peter-cnudde-yahoo-big-data.

[125] Hanchuan Peng, Fuhui Long, and Chris Ding.
Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy.
IEEE transactions on pattern analysis and machine intelligence, 27(8):1226–38, aug 2005.
ISSN 0162-8828.
doi: 10.1109/TPAMI.2005.159.
URL http://www.ncbi.nlm.nih.gov/pubmed/16119262.

[126] Daniel Peralta, Sara del Río, Sergio Ramírez-Gallego, Isaac Riguero, Jose M Benitez, and Francisco Herrera.
Evolutionary Feature Selection for Big Data Classification : A MapReduce Approach Evolutionary Feature Selection for Big Data Classification : A MapReduce Approach.
Mathematical Problems in Engineering, 2015(JANUARY), 2015.
doi: 10.1155/2015/246139.
URL http://sci2s.ugr.es/sites/default/files/2015-hindawi-peralta.pdf.

[127] Tossapol Pomsuwan and Alex A. Freitas.
Feature selection for the classification of longitudinal human ageing data.
In X Gottumukkala, R and Ning, X and Dong, G and Raghavan, V and Aluru, S and Karypis, G and Miele, L and Wu, editor, *IEEE International Conference on Data Mining Workshops, ICDMW*, volume 2017-Novem of *International Conference on Data Mining Workshops*, pages 739–746, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2017. IEEE; IEEE Comp Soc; Cisco; Citi, IEEE.
ISBN 9781538614808.
doi: 10.1109/ICDMW.2017.102.

[128] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery.
Numerical recipes in C, volume 2.
Cambridge Univ Press, 1982.

[129] J R Quinlan.
Induction of Decision Trees.
Mach. Learn., 1(1):81–106, mar 1986.

ISSN 0885-6125.
doi: 10.1023/A:1022643204877.
URL http://dx.doi.org/10.1023/A:1022643204877.

[130] J Ross Quinlan.
C4.5: Programs for Machine Learning, volume 1.
1992.
ISBN 1558602380.
doi: 10.1016/S0019-9958(62)90649-6.
URL http://portal.acm.org/citation.cfm?id=152181{%}5Cnhttp://www.amazon.com/C4-5-Programs-Machine-Learning-Kaufmann/dp/1558602380.

[131] Lawrence R Rabiner and Biing-Hwang Juang.
Fundamentals of speech recognition.
1993.

[132] Sergio Ramírez-Gallego, Iago Lastra, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Francisco Herrera, and Amparo Alonso-Betanzos.
Fast-mRMR: Fast Minimum Redundancy Maximum Relevance Algorithm for High-Dimensional Big Data.
International Journal of Intelligent Systems, 32(2):134–152, feb 2017.
ISSN 08848173.
doi: 10.1002/int.21833.
URL http://doi.wiley.com/10.1002/int.21833.

[133] Oscar Reyes, Carlos Morell, and Sebastián Ventura.
Scalable extensions of the ReliefF algorithm for weighting and selecting features on the multi-label learning context.
Neurocomputing, 161:168–182, 2015.
ISSN 09252312.
doi: 10.1016/j.neucom.2015.02.045.

[134] Irina Rish.
An empirical study of the naive Bayes classifier.
In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM, 2001.

[135] Marko Robnik-Šikonja and Igor Kononenko.
Theoretical and Empirical Analysis of ReliefF and RReliefF.
Machine Learning, 53(1/2):23–69, 2003.
ISSN 08856125.

doi: 10.1023/A:1025667309714.
URL http://link.springer.com/10.1023/A:1025667309714.

[136] Robert Rose.
Defining analytics: a conceptual framework.
ORMS-Today, 43(3), 2016.
URL https://www.informs.org/ORMS-Today/Public-Articles/June-Volume-43-Number-3/Defining-analytics-a-conceptual-framework.

[137] Peter Sadowski, Pierre Baldi, and Daniel Whiteson.
Searching for Higgs Boson Decay Modes with Deep Learning.
Advances in Neural Information Processing Systems 27 (Proceedings of NIPS), pages 1–9, 2014.
ISSN 10495258.

[138] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga.
A review of feature selection techniques in.
Bioinformatics, 23(19):2507–2517, 2007.
ISSN 1367-4803, 1460-2059.
doi: 10.1093/bioinformatics/btm344.

[139] Arthur L Samuel.
Some Studies in Machine Learning Using the Game of Checkers.
3(3):535–554, 1959.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.2254{&}rep=rep1{&}type=pdf.

[140] N Sánchez-Maroño, A Alonso-Betanzos, and M Tombilla-Snaromán.
Filter methods for feature selection–a comparative study.
Intelligent Data Engineering and Automated Learning - IDEAL 2007, pages 178–187, 2007.
doi: 10.1007/978-3-540-77226-2.
URL http://link.springer.com/chapter/10.1007/978-3-540-77226-2{_}19.

[141] Juwei Shi, Yunjie Qiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold Reinwald, and Fatma Özcan.
Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics.
Proc. VLDB Endow., 8(13):2110–2121, sep 2015.
ISSN 2150-8097.
doi: 10.14778/2831360.2831365.
URL http://dx.doi.org/10.14778/2831360.2831365.

[142] Jorge Silva, Ana Aguiar, and Fernando Silva.

Parallel Asynchronous Strategies for the Execution of Feature Selection Algorithms.
International Journal of Parallel Programming, pages 1–32, feb 2017.
ISSN 0885-7458.
doi: 10.1007/s10766-017-0493-2.
URL http://link.springer.com/10.1007/s10766-017-0493-2.

[143] Natalia Silvis-Cividjian.
Pervasive Computing.
Undergraduate Topics in Computer Science, 2017.

[144] Surender Singh and Ashutosh Kumar Singh.
Web-Spam Features Selection Using CFS-PSO.
Procedia Computer Science, 125:568–575, jan 2018.
ISSN 1877-0509.
doi: 10.1016/J.PROCS.2017.12.073.
URL https://www.sciencedirect.com/science/article/pii/S1877050917328375.

[145] Omar S Soliman and Aliaa Rassem.
Correlation Based Feature Selection Using Quantum Bio Inspired Estimation of Distribution Algorithm.
In Chattrakul Sombattheera, Nguyen Kim Loi, Rajeev Wankar, and Tho Quan, editors, *Multi-disciplinary Trends in Artificial Intelligence*, number Ml, pages 318–329, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
ISBN 978-3-642-35455-7.

[146] Thomas Sterling, Matthew Anderson, and Maciej Brodowicz.
High Performance Computing.
Morgan Kaufmann, 2018.
ISBN 9780124201583.
doi: 10.1016/C2013-0-09704-6.
URL https://linkinghub.elsevier.com/retrieve/pii/C20130097046.

[147] Matthew E Stokes and Shyam Visweswaran.
Application of a spatially-weighted Relief algorithm for ranking genetic predictors of disease.
BioData mining, 5(1):20, 2012.

[148] Yijun Sun.
Iterative RELIEF for feature weighting: algorithms, theories, and applications.
IEEE transactions on pattern analysis and machine intelligence, 29(6):1035–1051, 2007.

[149] Murdoch TB and Detsky AS.

The inevitable application of big data to health care.
JAMA, 309(13):1351–1352, apr 2013.
ISSN 0098-7484.
URL http://dx.doi.org/10.1001/jama.2013.393.

[150] Grigorios Tsoumakas, Ioannis Katakis, and An Overview.
Multi-Label Classification : An Overview.
International Journal of Data Warehousing and Mining, 3(September):1–13, 2007.
ISSN 15483924.
doi: 10.1109/ICWAPR.2007.4421677.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9401{&}rep=rep1{&}type=pdf.

[151] John W. Tukey.
The Future of Data Analysis.
The Annals of Mathematical Statistics, 33(1):1–67, 1962.
ISSN 0003-4851.
doi: 10.1214/aoms/1177704711.
URL http://projecteuclid.org/euclid.aoms/1177704711.

[152] Ryan J. Urbanowicz, Melissa Meeker, William La Cava, Randal S. Olson, and Jason H. Moore.
Relief-based feature selection: Introduction and review.
Journal of Biomedical Informatics, 85:189–203, sep 2018.
ISSN 1532-0464.
doi: 10.1016/J.JBI.2018.07.014.
URL https://www.sciencedirect.com/science/article/pii/S1532046418301400.

[153] Ryan J Urbanowicz, Randal S Olson, Peter Schmitt, Melissa Meeker, and Jason H Moore.
Benchmarking relief-based feature selection methods for bioinformatics data mining.
Journal of biomedical informatics, 85:168–188, 2018.

[154] Mark van Rijmenam.
Big Data at Walmart is All About Big Numbers; 40 Petabytes a Day!, 2015.
URL https://datafloq.com/read/big-data-walmart-big-numbers-40-petabytes/1175.

[155] Maarten van Steen and Andrew Tanenbaum.
Distributed Systems.
CreateSpace Independent Publishing Platform, 2017.
ISBN 978-1543057386.

[156] V Vapnik.
The Nature of Statistical Learning Theory, 1995.

[157] Yong Wang, Wenlong Ke, and Xiaoling Tao.
A Feature Selection Method for Large-Scale Network Traffic Classification Based on Spark.
Information, 7(1):6, feb 2016.
ISSN 2078-2489.
doi: 10.3390/info7010006.
URL http://www.mdpi.com/2078-2489/7/1/6.

[158] Ian H. Witten, Eibe Frank, and Mark a. Hall.
Data Mining: Practical Machine Learning Tools and Techniques, Third Edition, volume 54.
2011.
ISBN 9780123748560.
doi: 10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C.
URL http://www.cs.waikato.ac.nz/{~}ml/weka/book.html{%}5Cnhttp://www.amazon.com/Data-Mining-Practical-Techniques-Management/dp/0123748569.

[159] Rui Xu and D Wunsch.
Survey of clustering algorithms.
IEEE Transactions on Neural Networks, 16(3):645–678, may 2005.
ISSN 1045-9227.
doi: 10.1109/TNN.2005.845141.

[160] Lei Yu and Huan Liu.
Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution.
International Conference on Machine Learning (ICML), pages 1–8, 2003.
ISSN 01469592.
doi: citeulike-article-id:3398512.
URL http://www.aaai.org/Papers/ICML/2003/ICML03-111.pdf.

[161] Amelia Zafra, Mykola Pechenizkiy, and Sebastián Ventura.
ReliefF-MI: An extension of ReliefF to multiple instance learning.
Neurocomputing, 75(1):210–218, 2012.
ISSN 09252312.
doi: 10.1016/j.neucom.2011.03.052.

[162] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica.
Spark : Cluster Computing with Working Sets.
HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, page 10, 2010.

ISSN 03642348.
doi: 10.1007/s00256-009-0861-0.

[163] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, and Ankur Dave.
Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.
NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, page 2, 2012.
ISSN 00221112.
doi: 10.1111/j.1095-8649.2005.00662.x.
URL https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf.

[164] Yi Zhang, Chris Ding, and Tao Li.
Gene selection algorithm by combining reliefF and mRMR.
BMC Genomics, 9(Suppl 2):S27, 2008.
ISSN 1471-2164.
doi: 10.1186/1471-2164-9-S2-S27.
URL http://bmcgenomics.biomedcentral.com/articles/10.1186/1471-2164-9-S2-S27.

[165] Liang Zhao, Zhikui Chen, Yueming Hu, Geyong Min, and Zhaohua Jiang.
Distributed Feature Selection for Efficient Economic Big Data Analysis.
IEEE Transactions on Big Data, 4(2):164–176, jun 2018.
ISSN 2332-7790.
doi: 10.1109/TBDATA.2016.2601934.
URL https://ieeexplore.ieee.org/document/7549067/.

[166] Zheng Zhao and Huan Liu.
Searching for interacting features.
IJCAI International Joint Conference on Artificial Intelligence, pages 1156–1161, 2007.
ISSN 10450823.
doi: 10.3233/IDA-2009-0364.

[167] Zheng Zhao, Ruiwen Zhang, James Cox, David Duling, and Warren Sarle.
Massively parallel feature selection: an approach based on variance preservation.
Machine Learning, 92(1):195–220, jul 2013.
ISSN 0885-6125.
doi: 10.1007/s10994-013-5373-4.
URL http://link.springer.com/10.1007/s10994-013-5373-4.

[168] Zhi-hua Zhou.
Multi-instance learning: a survey.
AI Lab, Department of Computer Science and Technology, pages 1–31, 2004.
ISSN 03029743.
doi: 10.1109/CVPR.2012.6247772.
URL http://cs.nju.cn/zhouzh/zhouzh.files/publication/techrep04.pdf.