Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Parallel-FST: Aceleración de algoritmos de selección de características mediante computación paralela

Estudante: Bieito Beceiro Fernández
Dirección: Jorge González Domínguez
Dirección: Juan Touriño Domínguez

A Coruña, xuño de 2020.

A todas aquelas persoas que me motivaron a seguir adiante.

Agradecementos

Á miña familia e aos meus amigos e amigas, por comprender o que esta etapa significaba para min e por apoiarme sempre, ata nos momentos máis complicados.

A todos os mestres que, con paixón polo ensino, fostes capaces de transmitir os vosos coñecementos de forma amena. En especial a Jorge e a Juan, por confiar en min para a realización deste proxecto e por axudarme e guiarme dende o primeiro momento.

Moitas grazas a todos, sen vós isto non sería posible.

Bieito

Resumo

Na actualidade estase a producir un auxe da produción e consumo de grandes cantidades de información (*big data*), que deben procesarse e prepararse para o seu posterior uso. Entre as ferramentas que se utilizan para analizar estes datos atópanse as de aprendizaxe máquina (*machine learning*), o que constitúe outro campo de investigación que gañou importancia nos últimos anos. A pesar dos seus bos resultados, as técnicas de aprendizaxe automática contan cun custo computacional alto, que se incrementa notablemente ao aumentar a cantidade de datos a procesar. Para reducir a dimensionalidade destes datos, existen algoritmos de selección de características que, a través de modelos matemáticos, son capaces de eliminar información redundante e innecesaria. Porén, a selección de características tamén é un proceso custoso, pero que pode acelerarse adaptando os algoritmos e técnicas xa existentes para o seu uso en sistemas de computación paralela (coñecidos como HPC polas súas siglas en inglés).

Ao longo dos últimos anos xurdiron moitos traballos de investigación centrados no desenvolvemento de diferentes métodos de selección de características, cada un aplicando uns criterios de cara á devandita selección. Polo xeral, estes criterios deben tentar maximizar a relevancia das características seleccionadas e minimizar a redundancia entre as mesmas, de forma que o subconxunto escollido represente da mellor forma posible ao dataset orixinal. Tamén existen estudos que traballan con varios destes métodos para atopar o grao de conformidade entre os mesmos, para buscar similitudes a nivel de estrutura ou con intención de determinar cal presenta un mellor comportamento en termos de precisión, estabilidade e flexibilidade ante datasets de certas propiedades. Para este tipo de estudos moitas veces é necesario o desenvolvemento de librarías que conteñan os métodos de selección de características a estudar, de forma que se poidan comparar os resultados. Este é o caso de FEAST, unha libraría que conta con oito métodos de selección de características baseada en información mutua.

Neste Traballo Fin de Grao desenvolveuse unha optimización de FEAST con técnicas paralelas, adaptando os seus métodos para que poidan ser executados e aproveiten as vantaxes dos sistemas HPC. As paralelizacións implementadas desenvolvéronse aplicando unha distribución da carga de traballo entre elementos de procesado. Dado que os sistemas HPC adoitan ser sistemas multinodo con nodos multinúcleo, esta nova versión aproveita as posibilidades que achegan ambos cunha aproximación híbrida baseada en MPI e tecnoloxías multifío. A estratexia aplicada en ambos niveis foi a descomposición de dominio, i.e. a distribución dos datos cos que traballa o programa para que cada elemento de procesado realice os cálculos sobre un anaco diferente. Deste xeito conseguiuse, por unha parte, reducir o tempo de cómputo; e por outra, posibilitar a análise de datasets de gran tamaño que exceden as limitacións de memoria dos sistemas habituais.

As probas de rendemento realizáronse nun clúster de 16 nodos, con 64GB de memoria e 16 núcleos por nodo (256 núcleos en total). Os resultados obtidos foron moi satisfactorios, xa que se acadaron unhas aceleracións de ata 229x para catro datasets representativos. A maiores, conseguiuse executar cada algoritmo cun dataset de 512GB de tamaño, o que non sería posible nun único nodo.

Abstract

Currently, there is a boom in the production and consumption of large amounts of information (big data), which must be processed and prepared for later use. Machine learning techniques are among the tools used to analyze this data. Therefore, it is another field of research that has gained importance in recent years. Despite their good results, machine learning techniques have a high computational cost, which is significantly increased as the amount of data to be processed grows. To reduce the dimensionality of this data, there are feature selection algorithms able to remove redundant and unnecessary information with the use of mathematical models. However, feature selection is also an expensive process, but it can be accelerated by adapting existing algorithms and techniques to be run in high performance computing systems (HPC).

In recent years, many research projects have been focused on the development of different methods for feature selection, which apply some specific criteria to this selection. Usually, these criteria should try to maximize the relevance of the selected features and minimize the redundancy between them, so that the chosen subset represents the original data set in the best possible way. There are also studies that take into account several of these methods to find the degree of conformity between them, to look for similarities at the structure level or to determine which one performs best in terms of precision, stability and flexibility when applied to data sets of certain properties. For this kind of research, the development of libraries with several feature selection methods to be studied is often necessary in order to compare their results. This is the case of FEAST, a library that presents eight feature selection methods based on mutual information.

In this work a parallelization of the FEAST library has been developed, adapting its methods so that they can be executed and take advantage of HPC systems. The implemented parallelizations were developed by applying a workload distribution among processing elements. Since HPC systems are often multinode systems with multicore nodes, this new version takes advantage of the possibilities that both offer with a hybrid approach based on MPI and multithreading technologies. The strategy applied at both levels was the domain decomposition, that is, the distribution of the data used in the program, so that each processing element performs the calculations on a different part. This way, it was possible, on the one hand, to reduce

execution times; and, on the other hand, to allow the analysis of large data sets that exceed memory limitations of common systems.

Performance tests were carried out on a 16-node cluster with 64GB of memory and 16 cores per node (256 total cores). The obtained results are very satisfactory, since accelerations of up to 229x were achieved for four representative data sets. In addition, every algorithm was able to analyze a 512GB dataset, which would not have been possible on a single node.

Palabras chave:

- Selección de Características
- Información Mutua
- Redución da Dimensionalidade
- Aprendizaxe Máquina
- MPI
- Comput. de Altas Prestacións
- Computación Paralela
- Big Data

Keywords:

- Feature Selection
- Mutual Information
- Dimension Reduction
- Machine Learning
- MPI
- High Performance Computing
- Parallel Computing
- Big Data

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

A o longo deste capítulo achégase unha pequena contextualización e motivación do Traballo Fin de Grao, seguidas da presentación dos obxectivos do mesmo, e dunha explicación da estrutura desta memoria.

1.1 Motivación

Na actualidade estamos asistindo á explosión do fenómeno *big data*, onde cada vez obtense e almacénase unha maior cantidade de datos en diferentes áreas como a bioloxía, bioinformática, marketing ou socioloxía. Porén, os datos só son útiles na medida en que podemos analizalos para extraer información interesante. O aumento da cantidade de datos retarda as análises (incrementando o seu custo), pero non sempre proporciona mellor información debido á existencia de datos redundantes e irrelevantes. Por isto, na actualidade é de vital importancia simplificar os conxuntos de datos eliminando aqueles que proporcionan información redundante ou directamente irrelevante, utilizando técnicas de aprendizaxe máquina, un campo en gran auxe dentro da Intelixencia Artificial.

O proceso de aprendizaxe máquina que selecciona só aquelas características que aportan información relevante denomínase "selección de características", e o seu uso é cada vez máis común en diferentes ámbitos como a bioinformática, a xenética ou a recuperación de información de redes sociais. Existen diferentes métodos de selección de características, cada un coas súas vantaxes e inconvenientes, de forma que os seus usuarios (investigadores, analistas) escollen o que máis se adecúa ás análises que queren realizar. O maior inconveniente de moitos deles é que requiren grandes recursos computacionais, de memoria e de disco ao traballar con grandes conxuntos de datos.

O emprego da computación paralela constitúe unha boa solución para reducir os tempos dos métodos de selección de características. Neste Traballo Fin de Grao desenvolveuse unha libraría que inclúe implementacións paralelas de diferentes métodos de selección de caracte-

rísticas de cara executarse en sistemas de memoria distribuída como clústeres ou supercomputadoras. Para isto, tomaranse como base os métodos da libraría secuencial FEAST [1], xa que é moi utilizada e referenciada na literatura. Esta libraría contén unha serie de métodos de selección de características baseados en información mutua, pero que, ao contar cunha complexidade relativamente elevada, non se adecúan ben en casos nos que a cantidade de datos aumenta de forma significativa.

1.2 Obxectivos

O principal obxectivo deste proxecto consiste no deseño e implementación dunha libraría paralela de métodos de selección de características, Parallel-FST (*Parallel Feature Selection Toolbox*). Esta libraría habilita o uso destes métodos en sistemas de computación de altas prestacións (coñecidos como HPC polas súas siglas en inglés, *High Performance Computing*), reducindo significativamente os seus tempos de execución. Partirase da libraría de referencia FEAST, que contén varios métodos secuenciais de selección de características baseados en información mutua. En concreto, trataranse todos os algoritmos incluídos en FEAST para a linguaxe C.

Os sistemas HPC obxectivo deste Traballo Fin de Grao son sistemas multinodo (clústeres e supercomputadoras) onde cada nodo é multinúcleo. Polo tanto, co obxecto de conseguir o mellor aproveitamento dos mesmos, os deseños paralelos dos algoritmos utilizarán técnicas de paralelismo explícitas, é dicir procesos que se comunican mediante paso de mensaxes, e implícitas, mediante fíos con memoria compartida. En concreto, utilizarase a libraría MPI e fíos de C++, respectivamente.

Unha vez rematado o desenvolvemento de Parallel-FST, procederase a realizar unha avaliación experimental dos algoritmos para diferentes conxuntos de datos representativos e diferentes configuracións de procesos MPI e fíos de C++, co obxectivo de coñecer as melloras que achegan o uso das técnicas de computación paralela. Finalmente, a libraría resultante liberarase para que poida ser utilizada por científicos e analistas.

1.3 Estrutura da memoria

Esta memoria comeza cun capítulo de introdución de conceptos previos sobre a selección de características. Posteriormente, no segundo capítulo, explícanse detalladamente as etapas de deseño e implementación da libraría paralela Parallel-FST. A continuación, móstrase o procedemento seguido para realizar as probas de rendemento, xunto coa interpretación dos resultados obtidos. O quinto capítulo aborda a planificación que se seguiu durante o Traballo Fin de Grao. Para finalizar, expóñense as principais conclusións e introdúcense algunhas ideas para traballo futuro.

Capítulo 2

Métodos para redución da dimensionalidade

Neste capítulo proporcionarase unha introdución á redución da dimensionalidade, incidindo nos métodos de selección de características baseados en información mutua, e profundizando posteriormente nos métodos propostos na libraría FEAST[1]. Como xa se mencionou con anterioridade, esta será a libraría base para este traballo xa que é moi completa (presenta moitos métodos de selección de características) e moi empregada pola comunidade científica (máis de 800 citas en Google Scholar).

2.1 Conceptos básicos

Antes de comezar coa materia de estudo, semella oportuno aclarar o significado dunha serie de termos que aparecerán repetidamente ao longo deste capítulo e, en xeral, en todo o documento.

- **Característica** (ou *feature*). Representa unha propiedade individual e medible dun fenómeno observado. Pode tomar valores cualitativos (e.g. cor de ollos) ou cuantitativos (e.g. altura).

- **Clase**. No campo da aprendizaxe supervisada, precísase especificar que característica cumpre a función do valor a predicir. Os problemas de clasificación adoitan utilizar un conxunto de valores finitos para representar os grupos nos que se divide a poboación de estudo. Neste caso, a característica a predicir identifica o grupo ao que pertence unha mostra, e recibe o nome de clase. Pola contra, cando o problema é de regresión, o que se tenta é predicir un valor continuo a partir das demais variables independentes (i.e. as outras características); neste caso non existe un nome definido para o valor predito,

polo que, para referise ao mesmo, empregaranse os termos "valor de saída" ou "variable dependente".

- **Mostra** (ou *sample*). Está relacionado co significado do termo no campo da estatística, onde a mostra é un subconxunto de casos da poboación de estudo. Neste caso, cunha mostra referímonos a un só caso (ou equivalentemente) ao elemento dun subconxunto de cardinal 1. Dende o punto de vista da representación dos datos, unha mostra sería un vector de valores para as características de estudo.

- **Conxunto de datos** (ou *dataset*). Un "*dataset*" é unha estrutura de datos que almacena a relación entre o subconxunto de mostras considerado cos valores de cada característica de estudo. Polo xeral, estas relacións represéntanse en forma de matriz, que, en función das propiedades de cada problema, pode almacenarse de xeito denso ou disperso. Nisto profundizarase máis adiante, cando se expliquen os formatos dos datos de entrada. Para ilustrar un caso práctico dun dataset, tomaremos como exemplo o da flor de Iris[1]. Este conxunto de datos reúne a información sobre catro características (largo e ancho de sépalo e pétalo, en centímetros) de 150 mostras, divididas en tres clases (*Setosa*, *Versicolour* e *Virginica*). Na Figura 2.1 pódense observar as oito primeiras mostras do conxunto, onde, por exemplo, a primeira representa unha flor do tipo *Setosa*, coas seguintes propiedades:

  - Largo do sépalo: 5.1cm
  - Ancho do sépalo: 3.5cm
  - Largo do pétalo: 1.4cm
  - Ancho do pétalo: 0.2cm
  - Clase: *Setosa*

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa

Figura 2.1: Oito primeiras mostras do dataset Iris

[1] Dispoñible no repositorio de *machine learning* da Universidade de California.[2]

- **Relevancia**. O termo relevancia fai referencia á importancia de algo. Isto non é algo obxectivo nin medible, polo que non se pode calcular un valor absoluto. Pero pódese tentar estimar para poder establecer relacións de orde en canto a se un evento é máis relevante ca outro. Dende o punto de vista da análise de datos, unha característica considérase máis relevante cando ten maior potencial para a resolución dun problema dado. Por exemplo, en clasificación, a característica de maior relevancia é a que máis información aporta sobre a clase.

- **Redundancia**. A redundancia é un indicador de repetitividade, é dicir, un evento considérase redundante cando a información que achega xa é coñecida. Ao igual que no caso da relevancia, é unha apreciación relativa e non medible, polo que se debe establecer un criterio para estimala. Un exemplo en análise de datos pode ser se engadimos a un dataset unha característica repetida, ou que sexa combinación lineal doutras xa presentes.

2.2 Redución da dimensionalidade

Como se viu anteriormente, unha mostra representa uns datos para unha instancia dunha poboación, e pódese representar como un vector de valores. Deste xeito, dende o punto de vista matemático, cada característica pódese interpretar como unha dimensión, e cada mostra como un punto no espazo definido por esas dimensións. Polo tanto, cando falamos de redución da dimensionalidade, referímonos á redución do número de características que se utilizan nun dataset para representar cada mostra.

Nun primeiro momento, pódese pensar que é unha boa idea contar coa maior cantidade de datos posible, pero isto non sempre é así, e incluso nalgúns casos ata se conseguen mellores resultados partindo de menos información. O obxectivo deste apartado é expoñer varios puntos polos que pode ser interesante reducir o tamaño dos datasets, e que se enumeran a continuación:

- Dende o punto de vista das computadoras. As técnicas de análise de datos e aprendizaxe máquina, polo xeral, utilizan métodos con complexidades computacionais altas. Por outra parte, para poder aplicar ditas operacións, os datos necesitan cargarse en memoria, a que, a pesar das capacidades do hardware actual, non sempre é suficiente. Polo tanto, un dos motivos principais polos que se precisan datasets de menor tamaño é posibilitar a execución dos algoritmos de análise requeridos, e non só posibilitala, senón que ademais a execución remate nun tempo razoable.

- Dende o punto de vista das persoas. O procesado de *big data* non sempre ten como consumidor un axente virtual. Moitas veces, o obxectivo é a simplificación da visualización

dos datos por parte de persoas. Por exemplo, as disciplinas de *business intelligence* tentan simplificar a presentación da información que recolle unha empresa para axudar a tomar decisións na dirección da mesma. Aínda que na actualidade comezan a aparecer novos dispositivos, polo xeral a visualización realízase en dispositivos planos (i.e. monitores, televisións, etc.), polo que a representación de información de alta dimensionalidade coas posibilidades que ofrecen estes dispositivos non é trivial. É dicir, neste caso, a redución do número de dimensións ten como obxectivo que as persoas poidan comprender de forma máis fácil o significado dos datos de interese.

- Dende o punto de vista da información. Cun maior número de características conséguese unha descrición máis precisa para cada individuo, e con máis mostras obtense unha mellor representación do espazo mostral a estudar. Por outra parte, se as características a extraer non se escollen con precaución, moitas delas poderían ter unha influencia negativa nos resultados dos algoritmos de aprendizaxe máquina. Por exemplo, unha característica redundante poderíalle dar máis peso a unhas variables de entrada do que realmente teñen; unha característica independente ao problema podería producir relacións sen sentido; e unha característica con moito ruído podería ocasionar perdas de precisión nos resultados. En canto ás mostras, cabe destacar que contar cun maior número delas permite coñecer mellor o rango de valores que pode tomar unha característica das observadas, permitindo obter unha idea máis precisa da cantidade de información que garda a mesma. En resumo, é necesario atopar un equilibrio entre o número de características (menor, para evitar datos irrelevantes) e o de mostras (maior, para obter unha mellor cobertura do espazo) para manter a maior cantidade de información e minimizar o espazo que se necesita para o almacenamento.

En [3] proporciónase unha clasificación das técnicas de redución de dimensionalidade según dous criterios: se os datos están etiquetados ou non (para un problema de aprendizaxe, se é supervisada ou non, respectivamente), e se se basean na selección ou na transformación das características. Dado que o primeiro criterio depende do problema que se necesite tratar e dos seus datos, centrarémonos nas diferenzas entre as categorías do segundo. Cabe destacar que calquera das dúas aproximacións tentan reducir a dimensionalidade, pero a costa de perda de información. De feito, isto producirase sempre que, sexa por eliminación directa ou pola perda producida nunha proxección, se eliminen datos. Sen dúbida, isto é algo a ter en conta, polo que sempre se debe tentar que esta perda sexa mínima.

2.2.1 Transformación de características

A idea na que se basea esta técnica consiste en aplicar algún tipo de transformación ás características, de xeito que se cree un novo subconxunto alternativo máis compacto e de menor

tamaño. Estas técnicas tamén se coñecen como proxección de características, xa que o proceso de transformación conleva a combinación de dúas ou máis características, o que equivale á operación matemática de proxección dun espazo de varias dimensións noutro menor. Porén, este tipo de procedementos, polo xeral, ocasionan a perda de significado das características, xa que as características do dataset resultante son combinacións das do orixinal. A continuación achégase unha descrición xeral das dúas técnicas máis utilizadas para transformación:

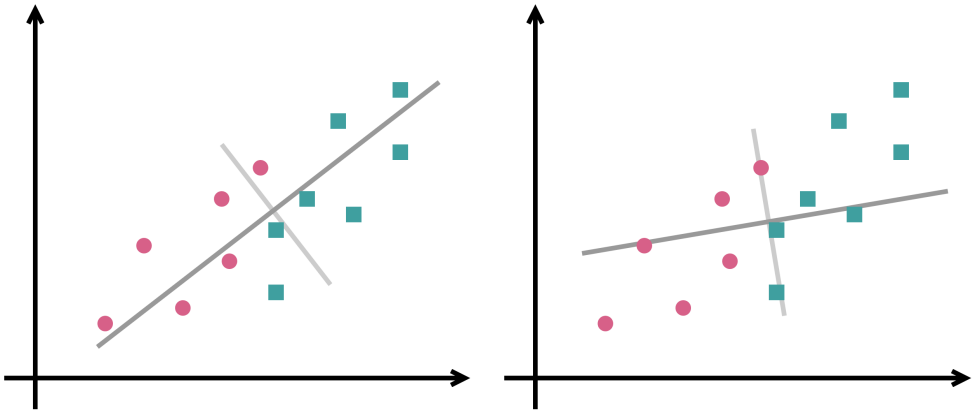
- *Principal Component Analysis (PCA).* A análise de compoñentes principais busca a proxección segundo a que os datos queden mellor representados en termos de mínimos cadrados. Para isto, constrúese unha transformación lineal para a que se escolle un sistema de coordenadas no que se captura a maior varianza do conxunto de datos no primeiro eixo, a seguinte maior varianza no seguinte eixo, e así sucesivamente. Tras realizar este procedemento, o número de características do dataset segue a ser o mesmo, pero é posible reducir a dimensionalidade eliminando as características que representan aos eixos de menor varianza, de forma que se minimiza a perda de información. Cabe destacar que este método non ten en conta o concepto de "clase", polo que adoita dar mellores resultados para conxuntos de datos en problemas de aprendizaxe non supervisada.

- *Linear Discriminant Analysis (LDA).* A análise discriminante lineal tamén tenta atopar unha proxección nun novo sistema de coordenadas. Porén, este método precisa coñecer a clase das mostras, xa que se busca un resultado que mellore a separabilidade das mesmas, en lugar de procurar maximizar a varianza unicamente.

Na Figura 2.2 pódese observar a diferenza dos dous métodos comentados. No caso da análise de compoñentes principais, o novo eixo máis longo é o que achega a maior varianza de todos os datos, polo que é sobre o que se realiza a proxección. Porén, nesta nova proxección as clases aparecen mesturadas, polo que a separación non sería precisa. En cambio, no caso da análise discriminante lineal, a división entre as clases está mellor definida na nova proxección.

2.2.2 Selección de características

Este Traballo Fin de Grao céntrase nas técnicas de selección de características. Neste caso, o procedemento que se segue para a busca do novo conxunto de datos de menor tamaño baséase en atopar o "mellor" subconxunto de características do dataset orixinal. Por este motivo, o resultado do procedemento é interpretable, xa que as características non eliminadas manteñen o significado que posuían inicialmente. Polo xeral, estas técnicas utilízanse en combinación co adestramento de modelos de aprendizaxe supervisada, polo que o obxectivo



(a) Principal Component Analysis (b) Linear Discriminant Analysis

Figura 2.2: Resultado de aplicar PCA ou LDA sobre os mesmos datos

é seleccionar aquelas características que constrúen o modelo de mellor precisión. Por outra banda, na aprendizaxe non supervisada non se pode coñecer, a priori, que criterio define o "mellor" subconxunto, polo que a selección de características para este tipo de aprendizaxe é unha área menos explorada.

As técnicas de selección constan, en xeral, dunha estratexia de busca para a exploración dos subconxuntos de características, que inclúe algún método de xeración de candidatos e tamén algún criterio para avalialos e poder comparalos. En función de como se realice o proceso de avaliación, distínguense as seguintes tres categorías:

- *Filters.* As aproximacións de filtrado aplícanse con anterioridade ao algoritmo de aprendizaxe, con total independencia do mesmo. Polo tanto, é necesario atopar unha métrica que permita predicir aquelas características que darán mellores resultados, é dicir, aquelas coa maior relevancia do dataset. Tres das métricas máis comúns son *chi-cadrado*, *odds ratio* e *ganancia de información*. Como se pode ver na Figura 2.3, unha vez se calculan as puntuacións de cada característica, o seguinte paso é aplicar o filtro para seleccionar as máis importantes. Para isto, existen varias aproximacións, como coller as n mellores, ou coller todas as que superan unha determinada puntuación.

- *Wrappers.* Este tipo de aproximacións "envolven" ao algoritmo de aprendizaxe, como

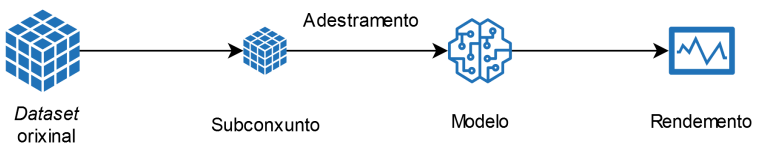


Figura 2.3: Esquema dos *filters*

se mostra na Figura 2.4. Isto é, tras propoñer un subconxunto, adéstrase o clasificador con ese subconxunto, e a calidade do mesmo dependerá da precisión do modelo. O principal problema desta aproximación é a súa natureza exponencial (decidir para cada característica se se escolle ou non), polo que existen varias estratexias para a busca:

- Busca cara adiante: comézase cun conxunto vacío, e vanse engadindo características unha a unha escollendo a mellor combinación anterior, ata que non se mellore o resultado.
- Busca cara atrás: comézase con todas as características e vanse eliminando unha a unha de cada mellor combinación previa, ata que non se mellore o resultado.
- Busca xenética: utilízase un algoritmo xenético para buscar por todo o espazo de solucións. Cada estado defínese por unha máscara que indica se cada característica está ou non.

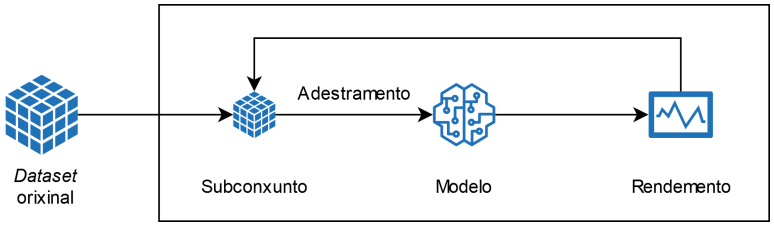


Figura 2.4: Esquema dos *wrappers*

- *Embeddeds.* As aproximacións embebidas aplican a selección como parte do propio algoritmo de aprendizaxe, como se representa na Figura 2.5. Un dos exemplos máis destacables son os algoritmos de construción de árbores de decisión, que para cada nodo deben escoller cal é o mellor criterio de decisión para seguir por unha rama ou outra (i.e. seleccionar que característica, ou combinación de varias, permite tomar unha mellor decisión sobre como clasificar o patrón).

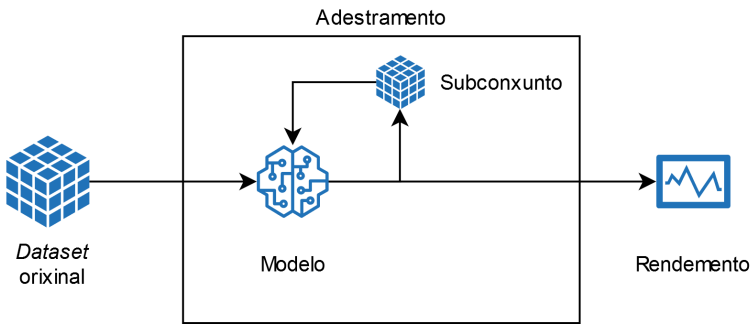


Figura 2.5: Esquema dos *embeddeds*

Cabe mencionar que os algoritmos contemplados neste proxecto pertencen á categoría de filtrado con criterios de información mutua, polo que se tratarán con maior detalle na seguinte sección.

2.3 Teoría da información

Neste Traballo Fin de Grao procúrase que o subconxunto resultante da fase de filtrado manteña a maior cantidade de información posible. Polo tanto, en primeiro lugar, é importante explicar uns conceptos previos para comprender o que proporcionan as medidas de información.

Antes de introducir estes conceptos, cabe mencionar que as seguintes métricas de información traballan sobre datos discretos e non continuos, polo que cando os datasets conteñen valores do espazo real é necesario aplicar un proceso de discretización (i.e. transformación dos datos continuos a discretos). Aínda que existen varias aproximacións, implementouse a coñecida como "*binning*", permitindo ao usuario determinar o número de *bins*. O funcionamento consiste en particionar o rango dos valores que pode tomar a característica en n *bins*, e despois substituír cada valor polo índice do *bin* no que estea contido. A continuación amósase un exemplo simple do funcionamento do *binning* para unha característica arbitraria f con valores entre 0.6 e 3.1, empregando cinco *bins* (ver Táboa 2.1), o que xera a característica f' (ver Táboa 2.2):

1. Calcúlase a lonxitude do rango de f: $l_f = max(f) - min(f)$

$$l_f = 3.1 - 0.6 = 2.5$$

2. Calcúlase a lonxitude de cada *bin*: $l_{bin} = \frac{l_f}{nBins}$

$$l_{bin} = \frac{2.5}{5} = 0.5$$

3. Determínase o rango de cada *bin* como se pode ver na Táboa 2.1: $bin_i = [min(f) + l_{bin} * i,\ min(f) + l_{bin} * (i + 1)]$

4. Constrúese a nova característica como os índices dos *bins* nos que están contidos os valores orixinais: $f'[k] = \{\, i \mid f[k] \in bin_i \,\}$

Os valores iniciais e finais das características tras aplicar o *binning* pódense ver na Táboa 2.2. Un código para o cómputo deste *binning* de datasets con datos non enteiros está incluído na libraría Parallel-FST.

Bin	0	1	2	3	4
Rango	[0.6, 1.1)	[1.1, 1.6)	[1.6, 2.1)	[2.1, 2.6)	[2.6, 3.1]

Táboa 2.1: Rango dos *bins* no exemplo

Característica	**Valores**				
f	1.4	0.6	0.9	3.1	2.8
f'	1	0	0	4	4

Táboa 2.2: Exemplo de *binning*

2.3.1 Cantidade de información

A cantidade de información que contén un evento aleatorio depende unicamente da súa probabilidade. É dicir, canto menor sexa a súa probabilidade, maior información terá, e viceversa. Pódese definir formalmente como

$$I(x) = -\log p(x) \tag{2.1}$$

onde $p(x)$ é a probabilidade do suceso x.

2.3.2 Entropía

A entropía é a unidade de información fundamental dunha variable aleatoria. Denótase por $H(X)$, e cuantifica a incerteza presente na distribución de X. Defínese como

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{2.2}$$

onde x denota un valor que a variable X pode tomar. Pódese interpretar como a incerteza presente na distribución de X. Polo tanto, é menor se a distribución está sesgada cara un evento particular, e máxima cando todos os eventos son equiprobables.

Seguindo as regras estándar da teoría da probabilidade, a entropía pode estar condicionada por outros eventos. Deste xeito, a entropía condicional de X dado Y calcúlase como

$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(x|y) \tag{2.3}$$

A entropía condicional pódese interpretar como a cantidade de incerteza conservada en X unha vez se coñece o resultado de Y.

2.3.3 Información mutua

Unha vez coñecida a entropía, podemos definir a información mutua, é dicir, a cantidade de información que comparten dúas variables. A información mutua entre X e Y calcúlase como

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= \sum_{x \in X} \sum_{y \in Y} p(xy) \log \frac{p(xy)}{p(x)p(y)} \end{aligned} \tag{2.4}$$

Xa que isto é a diferenza de dúas entropías, pódese interpretar como a cantidade de incerteza en X que se elimina ao coñecer Y. Ou, cunha expresión máis intuitiva, a cantidade de información que unha variable proporciona sobre outra.

Da mesma forma que a entropía, a información mutua tamén pode ser condicional. Isto pódese interpretar como a información que aínda comparten dúas variables despois de coñecerse unha terceira. A información mutua entre X e Y dado Z, calcúlase como

$$\begin{aligned} I(X;Y|Z) &= H(X|Z) - H(X|YZ) \\ &= \sum_{z \in Z} p(z) \sum_{x \in X} \sum_{y \in Y} p(xy|z) \log \frac{p(xy|z)}{p(x|z)p(y|z)} \end{aligned} \tag{2.5}$$

2.4 Selección de características baseada en información mutua

Nesta sección explicaranse os métodos de selección de características utilizados na libraría resultante deste Traballo Fin de Grao, que son os mesmos que están na amplamente empregada libraría FEAST [1]. Todos son métodos de filtrado que calculan as puntuacións das características con criterios de información.

Os métodos de filtrado defínense por un criterio J, que procura proporcionar unha medida de como de útil é unha característica ao usala nun clasificador. Tamén se coñece como "índice de relevancia" ou "puntuación" (*score* en inglés). Todos os métodos, a excepción do primeiro (o máis sinxelo), seguen unha aproximación iterativa para ir seleccionando unha a unha as l mellores características. Neles comézase creando un conxunto S vacío que ao final terá as características seleccionadas. En cada iteración calcúlase o J para todas as características que aínda non están en S e engádese a ese conxunto a característica con maior J.

2.4.1 MIM - Mutual Information Maximisation

Un cálculo intuitivo e simple da puntuación sería algún tipo de medida de correlación entre a característica e a etiqueta de clase. Revisando as medidas de información da sección anterior, pódese ver que a *información mutua* pode ser de utilidade. Deste xeito, para unha etiqueta de clase Y, o *score* dunha característica X_k, calcúlase como

$$J_{MIM}(X_k) = I(X_k; Y) \tag{2.6}$$

Esta heurística aparece moitas veces na literatura, por exemplo en [4]. Para utilizar esta medida, unicamente hai que calcular as puntuacións para cada característica e logo escoller as l mellores.

A principal desvantaxe deste método é que asume independencia entre as características, o que pode introducir redundancia nos resultados. Por exemplo, se a característica con maior correlación coa clase aparece repetida no dataset orixinal, escollerase dúas veces para o resultante. Porén, existe un caso en particular para o que isto non é un problema: cando só hai que seleccionar unha característica. Neste caso particular a solución de MIM permitiría construir o clasificador óptimo (cunha soa entrada) ao escoller a característica que máis información achega sobre a clase. Aínda que isto poida parecer un exemplo pouco común, o certo é que se dá en cada proceso de selección de características ao escoller a primeira das l a seleccionar, e polo tanto pódese utilizar con este fin en combinación con todos os demais métodos.

2.4.2 CondMI - Conditional Mutual Information

O criterio CondMI é unha optimización derivada a partir da formulación do problema de selección de características como un problema de probabilidade condicional [1]. Calcúlase a puntuación para unha característica X_k, unha clase Y e un conxunto de características seleccionadas S como

$$J_{CondMI}(X_k) = I(X_k; Y|S) \tag{2.7}$$

Isto é, a información mutua entre a característica e a clase, condicionada polas características escollidas ata o momento.

2.4.3 Criterios contidos no espazo BetaGamma

MIFS - Mutual Information Feature Selection

Este criterio preséntase en [5], e introduce melloras respecto a MIM para a redución da redundancia. A puntuación dunha característica X_k, para unha clase Y e un conxunto de características xa seleccionadas S, calcúlase como

$$J_{MIFS}(X_k) = I(X_k;Y) - \beta \sum_{X_j \in S} I(X_k;X_j) \tag{2.8}$$

onde β é un parámetro definido polo usuario que se pode interpretar como a disconformidade coa suposición de independencia entre características. É dicir, cando tende a 0, supón que calquera información mutua $I(X_k;X_j)$ é espúrea e producida por ruído, e cando tende a 1, supón que a información mutua se debe ter totalmente en conta para evitar a redundancia.

Como se pode observar, este criterio toma inicialmente a mesma medida de relevancia que MIM, isto é, a información mutua entre a característica e a clase. Porén, para paliar o problema da redundancia, introduce unha compoñente negativa, que indica a cantidade de información mutua entre a característica a puntuar e as xa seleccionadas. Cabe destacar que cando $\beta = 0$ o criterio é equivalente a MIM.

CIFE - Conditional Infomax Feature Extraction

Este criterio foi proposto en [6] e pódese obter a partir de varias transformacións de CondMI. A puntuación dunha característica X_k, para unha clase Y e un conxunto de características xa seleccionadas S, calcúlase como

$$J_{CIFE}(X_k) = I(X_k;Y) - \sum_{X_j \in S} I(X_k;X_j) + \sum_{X_j \in S} I(X_k;X_j|Y) \tag{2.9}$$

Pódense identificar tres termos principais: $I(X_k;Y)$ é a información mutua coa clase, polo tanto, indica relevancia; $\sum I(X_k;X_j)$ é a información mutua coas características xa seleccionadas, isto é, a redundancia; e $\sum I(X_k;X_j|Y)$ é a información mutua coas características xa seleccionadas e condicionada pola clase, o que se coñece como redundancia condicional. A maiores, este criterio considérase como un criterio "raíz", xa que se poden derivar outros moitos a partir del. Por exemplo, supoñendo que para todas as caraterísticas i, j temos $p(x_i x_j|y) = p(x_i|y)p(x_j|y)$ (i.e. as características son, en par, condicionalmente independentes coa clase), o último termo $\sum I(X_k;X_j|Y)$ vólvese 0, co que CIFE é equivalente a $\text{MIFS}_{\beta=1}$.

Espazo BetaGamma

Como se pode observar, os criterios MIFS e CIFE comparten unha forma común, polo que se pode imaxinar un espazo de dúas dimensións que permite expresar os criterios anteriores como combinación lineal de termos de medida de información. É dicir, con dous parámetros podemos especificar a importancia que lle damos aos termos de redundancia (β) e redundancia condicional (γ).

O espazo proposto recibe o nome de "BetaGamma" (dado polos dous parámetros que o determinan), e parametriza os criterios como

$$J_{BetaGamma}(X_k) = I(X_k;Y) - \beta \sum_{X_j \in S} I(X_k;X_j) + \gamma \sum_{X_j \in S} I(X_k;X_j|Y) \tag{2.10}$$

co que se poden calcular varios criterios xa vistos cos seguintes valores para os parámetros:

- **MIM** $\beta = 0, \gamma = 0$
- **MIFS** $\beta \in [0,1], \gamma = 0$
- **CIFE** $\beta = 1, \gamma = 1$

Na Figura 2.6 pódese observar unha representación gráfica de dito espazo, e as posicións nas que se sitúan os criterios vistos. Tamén aparecen outros criterios que se verán máis adiante, xa que se desprazan polo espazo en función do tamaño do conxunto de características xa seleccionadas.

Por este motivo, os criterios MIFS e CIFE implementáronse nun mesmo método "BetaGamma", que permite especificar os valores para ambos parámetros. Por outra parte, aínda que MIM tamén se pode calcular con $\beta = 0$ e $\gamma = 0$, implementouse cun método propio, para evitar o cálculo de medidas de información que se anularán ao multiplicalas por cero.

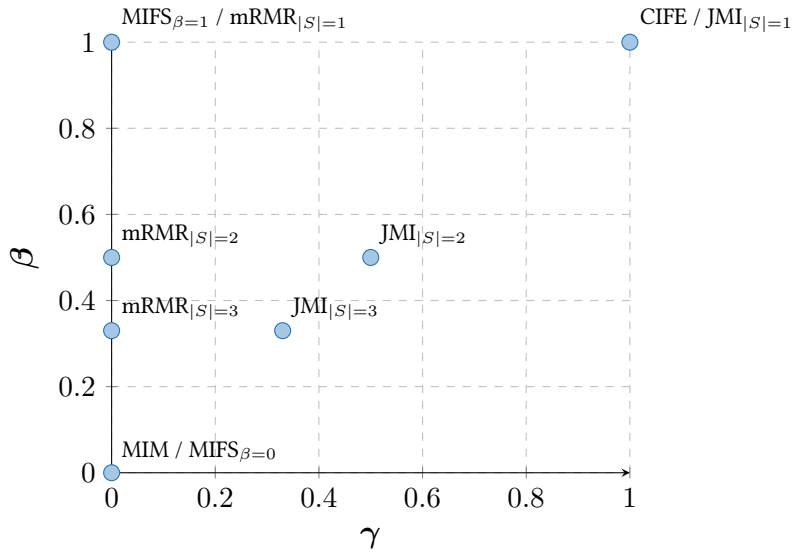


Figura 2.6: Espazo BetaGamma e criterios contidos

2.4.4 JMI - Joint Mutual Information

JMI é unha aproximación alternativa a MIFS presentada en [7] que, en lugar de tentar reducir a redundancia, tenta aumentar a información complementaria entre características. Calcúlase, para unha característica X_k, unha clase Y e un conxunto de características xa seleccionadas S como

$$J_{JMI}(X_k) = \sum_{X_j \in S} I(X_k X_j; Y) \tag{2.11}$$

É dicir, a suma de información entre a clase e unha variable aleatoria conxunta (*joint* en inglés) $X_k X_j$, definida unindo a candidata X_k con cada característica xa seleccionada. Deste xeito, este criterio baséase en que se debe incluír a característica candidata cando aporta información complementaria sobre as xa escollidas.

Se se reescribe como

$$J_{JMI}(X_k) = I(X_k; Y) - \frac{1}{|S|} \sum_{X_j \in S} [I(X_k; X_j) - I(X_k; X_j|Y)] \tag{2.12}$$

pódese observar que este criterio tamén aparece no espazo BetaGamma, no que se despraza dinamicamente en función do número de características seleccionadas ata o momento.

2.4.5 mRMR - Max-Relevance Min-Redundancy

mRMR é un método proposto en [8] que lle dá un valor ao parámetro β presente en MIFS en función do tamaño do subconxunto de características seleccionadas ata o momento. Así, calcula a puntuación para unha característica X_k, unha clase Y e o subconxunto de características seleccionadas S como

$$J_{mRMR}(X_k) = I(X_k; Y) - \frac{1}{|S|} \sum_{X_j \in S} I(X_k; X_j) \tag{2.13}$$

Pódese observar que omite o termo de redundancia condicional presente en CIFE. Por outra parte, o parámetro β é inversamente proporcional ao número de características seleccionadas, o que ocasiona que se vaia reducindo ao longo do proceso. Isto pódese interpretar como que ao ir aumentando o conxunto S, o criterio pon maior confianza en que as características son independentes entre si.

De igual forma que JMI, este criterio tamén está contido e desprázase polo espazo BetaGamma (neste caso, ao longo do eixo β).

2.4.6 CMIM - Conditional Mutual Information Maximisation

Todos os métodos presentados ata o momento comparten unha forma común. Isto non sucede con CMIM, proposto en [9], que utiliza operadores de orde para o cálculo da puntuación. En particular, calcula o *score* dunha característica X_k, para unha clase Y e un conxunto de características xa seleccionadas S, como

$$J_{CMIM}(X_k) = \min_{X_j \in S} [I(X_k; Y|X_j)] \tag{2.14}$$

que se pode reescribir como

$$J_{CMIM}(X_k) = I(X_k; Y) - \max_{X_j \in S} [I(X_k; X_j) - I(X_k; X_j|Y)] \tag{2.15}$$

O uso dos operadores de orde dificulta unha interpretación probabilística, pero pódese ver que o criterio outorga menores puntuacións a aquelas características con maior redundancia.

2.4.7 ICAP - Interaction Capping

Este criterio foi proposto en [10], e tamén utiliza operadores de orde. A puntuación dunha característica X_k, para unha clase Y e un conxunto de características xa seleccionadas S, calcúlase como

$$J_{ICAP}(X_k) = I(X_k; Y) - \sum_{X_j \in S} \max [0, \{I(X_k; X_j) - I(X_k; X_j|Y)\}] \tag{2.16}$$

Da mesma forma que CMIM, a interpretación probabilística non é simple, pero tamén se pode observar que tenta reducir a redundancia.

2.4.8 DISR - Double Input Symmetrical Relevance

Por último, DISR é unha modificación de JMI proposta en [11]. Calcúlase a puntuación para unha característica X_k, unha clase Y e un conxunto de características seleccionadas S como

$$J_{DISR}(X_k) = \sum_{X_j \in S} \frac{I(X_k X_j; Y)}{H(X_k X_j Y)} \tag{2.17}$$

Como se pode observar, introdúcese un termo de normalización, o que rompe a conexión teórica cunha función de probabilidade.

2.4.9 Versións *weighted*

Cabe destacar que varios dos métodos (MIM, JMI, CMIM, DISR e CondMI) contan con versións *weighted* ou ponderadas. Estas variacións permiten especificar un valor de relevancia para cada evento, de forma que isto se reflicte nos resultados do cálculo da entropía ou información mutua. Por exemplo, o cálculo da información mutua de dúas variables modificaríase da seguinte forma, onde $w(x, y)$ é o peso da co-ocorrencia de dous valores de cada variable X e Y

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} w(xy)p(xy) \log \frac{p(xy)}{p(x)p(y)} \tag{2.18}$$

Isto permite introducir coñecemento e dar significado a certas mostras, xa que a información mutua é totalmente independente das relacións de orde que se poidan establecer entre os valores que toma unha característica. De feito, unicamente depende das coincidencias que se dan entre as mesmas, nunca dos propios valores.

Capítulo 3

Deseño e implementación paralela

Este capítulo describirá o procedemento seguido para a obtención do sistema final (Parallel-FST), explicando a arquitectura obxectivo, solucións propostas en traballos relacionados, estrutura do programa e solucións a problemas atopados. O código está almacenado no repositorio de Gitlab da Facultade de Informática[1], e como unha das metas deste proxecto é que o produto resultante estea á disposición da comunidade científica, o mesmo liberarase proximamente.

3.1 Arquitectura obxectivo

Os sistemas cos que contamos para executar programas con alta complexidade de cálculo ou que traballan con grandes volumes de datos non son computadoras de uso persoal, senón clústeres compostos de varios nodos. A arquitectura obxectivo deste Traballo Fin de Grao pódese definir como un sistema de memoria distribuída constituído por varios nodos interconectados a través dunha rede, cada un deles cun módulo de memoria e varios núcleos de procesamento (ver Figura 3.1). A computación paralela neste tipo de sistemas xeralmente adopta o esquema *Single Program Multiple Data (SPMD)*, é dicir, divide a carga de traballo en diferentes tarefas que se executan en múltiples CPUs de forma que todos os nodos e núcleos colaboran para acelerar os cálculos. A capacidade computacional do clúster dependerá de factores como o número de nodos, o número de núcleos por nodo, as características da rede ou a velocidade de transferencia da memoria.

A continuación veremos como se poden utilizar os sistemas destas características, ao profundizar un pouco máis na arquitectura. En primeiro lugar, un nodo é a unidade que equivale a unha computadora, é dicir, está composto de memoria, núcleos de procesamento, almacenamento e sistema de entrada e saída. Con isto podemos apreciar o primeiro nivel no que se pode distribuír a carga de traballo: cada nodo executará unha parte do programa xeral.

[1] https://git.fic.udc.es/bieito.beceiro.fernandez/Parallel-FST

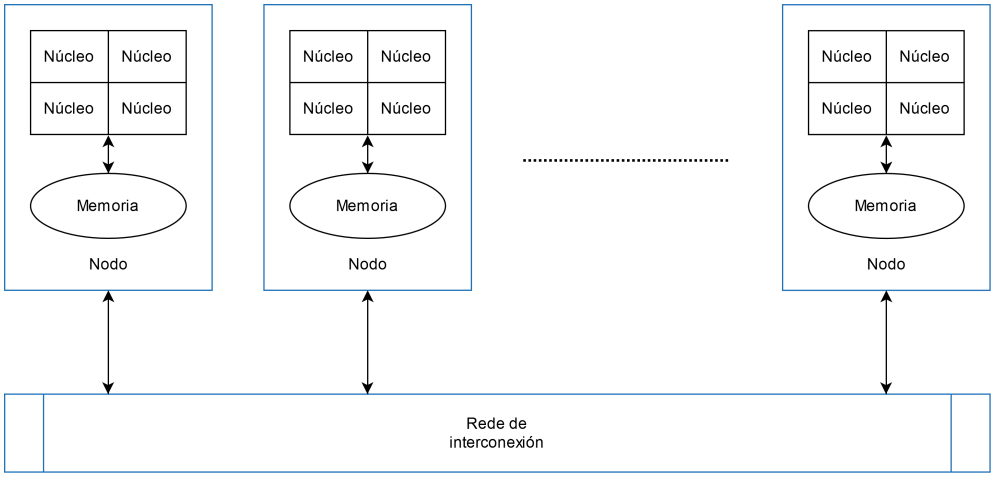


Figura 3.1: Abstracción dun sistema de memoria distribuída con varios núcleos e un módulo de memoria por nodo

Xa que cada nodo é unha unidade de memoria independente, a comunicación entre eles debe realizarse mediante paso de mensaxes. Este nivel de paralelismo denomínase explícito, xa que o programador debe especificar que datos se almacenan en cada memoria e cando se envían mensaxes entre un nodo e outro.

A máis baixo nivel aparece o paralelismo implícito, que é o que se dá dentro de cada nodo. Isto é posible xa que, como se comentou previamente, cada nodo está formado por varios núcleos de procesamento, o que permite repartir as tarefas entre eles e que fagan uso da memoria do nodo de forma compartida. A diferenza do paralelismo explícito, o implícito non precisa paso de mensaxes para a comunicación entre tarefas. Pola contra, no paralelismo implícito pode ser necesario o uso dalgún mecanismo auxiliar para evitar problemas de sincronización nos accesos á memoria compartida.

Tamén é común que nun sistema destas características haxa nodos con compoñentes adicionais, como poden ser as tarxetas gráficas (GPUs), que tamén se poden utilizar como elementos de cómputo. Aínda así, neste proxecto non se consideran este tipo de compoñentes, e o traballo centrarase en aplicar o paralelismo explícito e implícito de forma conxunta para explotar tódalas CPUs do clúster.

3.1.1 MPI

MPI (*Message Passing Interface*)[2] é unha especificación portable, eficiente e flexible para o desenvolvemento de aplicacións paralelas mediante paso de mensaxes. De feito, hoxe en día considérase como o estándar de facto para paso de mensaxes. É común referirse a MPI como unha libraría con funcións para comunicacións entre procesos. Pero isto non é así realmente

[2] https://www.mpi-forum.org/

xa que MPI é a especificación estándar que unha libraría debe adoptar para implementar o seu protocolo de paso de mensaxes. É máis, existen varias implementacións tanto propietarias como de código aberto, como poden ser OpenMPI[3] ou MPICH[4].

A especificación MPI permite traballar con múltiples elementos de procesado coordinados por paso de mensaxes. Cada elemento de procesado (denominado proceso) ten o seu propio espazo de memoria. Cando comeza a execución dun programa MPI créanse os procesos solicitados, que poden comunicarse mediante mensaxes para así intercambiar datos. O custo destas comunicacións depende das características do hardware, especialmente aquelas relacionadas coa rede de interconexión. A continuación explícanse varias das funcións MPI utilizadas neste Traballo Fin de Grao:

- **MPI_Init, MPI_Finalize e MPI_Abort** As dúas primeiras marcan o inicio e o fin do programa que utiliza algunha implementación da libraría. Empréganse para crear e eliminar, respectivamente, todas as estruturas de datos que necesita o programa MPI para poder enviar mensaxes entre todos os procesos. Pola súa parte, *MPI_Abort* permite realizar unha finalización abrupta de todos os procesos.

- **MPI_Comm_rank e MPI_Comm_size** Xeralmente utilízanse xuntas, xa que permiten coñecer o identificador do proceso actual, e o número de procesos totais, respectivamente.

- **MPI_Send e MPI_Recv** Aínda que estas dúas funcións non se utilizaron directamente no desenvolvemento, constitúen a base das comunicacións con MPI, xa que permiten a transferencia de datos máis básica entre dous procesos. O seu uso móstrase na Figura 3.2. Utilízanse de forma conxunta, de forma que o proceso que envía executa *MPI_Send* e o receptor *MPI_Recv*. Ambas funcións son bloqueantes, é dicir, a execución de cada proceso detense ata que remata a transferencia. Para comunicacións non bloqueantes existen as alternativas **MPI_Isend** e **MPI_Irecv**, respectivamente.

Figura 3.2: Transferencia simple con Send e Recv

- **MPI_Bcast** Realiza unha comunicación colectiva, enviando unha certa cantidade de datos dende un a todos os procesos, de forma que ao rematar todos contan coa mesma información. Na Figura 3.3 amósase unha abstracción do que significa esta operación. A

[3] https://www.open-mpi.org/
[4] https://www.mpich.org/

nivel de implementación interna, unha aproximación simple é que o proceso raíz envíe os datos aos demais, un por un, o que é moi ineficiente. Polo xeral, as implementacións máis utilizadas están optimizadas cunha estrutura en árbore, na que cada proceso que xa ten os datos contribúe á transferencia, enviándoos a outro proceso, como se pode observar na Figura 3.4.

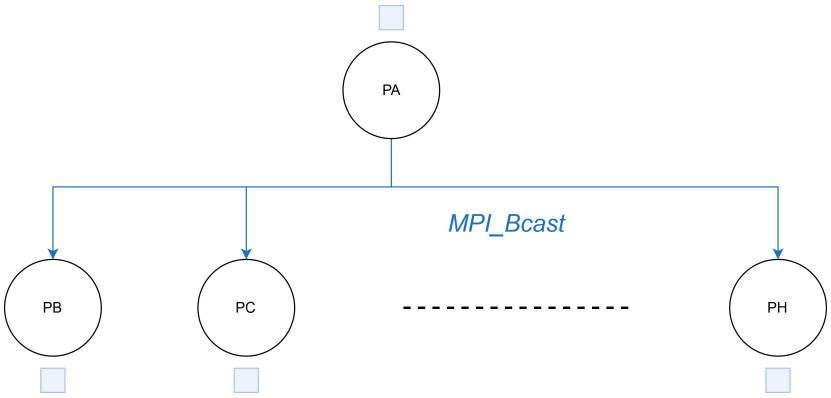


Figura 3.3: Transferencia con Bcast

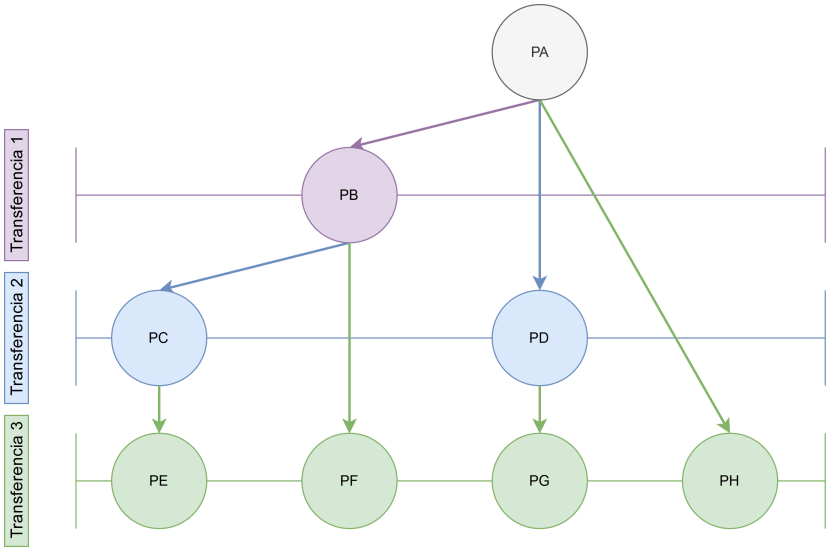


Figura 3.4: Implementación en árbore para Bcast

- **MPI_Scatterv** Para comprender a forma de operar desta función, partiremos da súa versión básica *MPI_Scatter*. Esta comunicación é similar a Bcast, pero a idea non é enviar a mesma información a todos os procesos, senón repartir un buffer de datos equitativamente entre eles. Este funcionamento pódese ver na Figura 3.5. Pola súa parte, *MPI_Scatterv* realiza esta mesma función, pero ten a particularidade de que permite de-

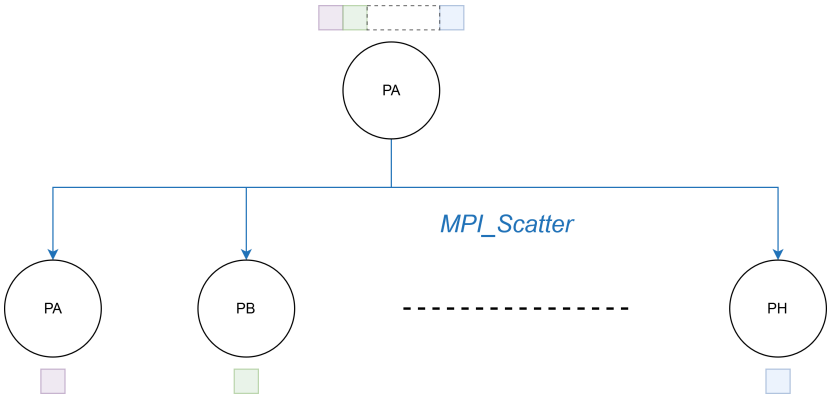


Figura 3.5: Funcionamento Scatter

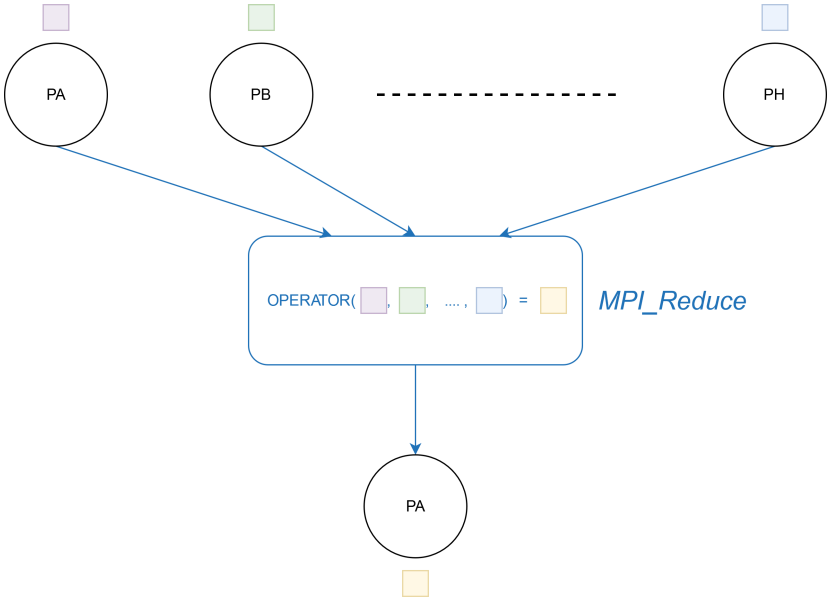


Figura 3.6: Funcionamento Reduce

finir a cantidade de datos que ten que recibir cada proceso (que non ten por que ser igual para todos) e o desprazamento, é dicir, a posición no buffer orixinal na que comezan os datos que lle corresponden a cada proceso.

- **MPI_Allreduce** Ao igual que no caso anterior, *MPI_Allreduce* tamén é unha versión estendida doutra función máis básica. Neste caso, a función base é *MPI_Reduce*, que utiliza datos de todos os procesos para aplicar unha operación sobre eles, e devolve o resultado ao proceso raíz, como se pode observar na Figura 3.6. *MPI_Allreduce* equivale á execución de *MPI_Reduce* seguida de *MPI_Bcast*, de forma que, despois de aplicar a operación, todos os procesos teñen o resultado da mesma.

  Neste proxecto, como operador para esta función, utilizouse *MPI_MAXLOC*, que computa o máximo global para todos os datos, tendo en conta que estes datos teñen un índice asociado. É dicir:

$$maxloc\left[(u,i),(v,j)\right] = (w,k)$$
$$w = max(u,v)$$
$$k = \begin{cases} i & u > v \\ min(i,j) & u = v \\ j & u < v \end{cases}$$

  o que é de moita utilidade cando se quere coñecer información a maiores da resultante da operación. Neste caso, por exemplo, para coñecer o identificador do proceso que posúe o valor máximo.

3.1.2 Tecnoloxías multifío

Os fíos (ou *threads*) son instancias de execución paralela xestionadas polo sistema operativo. A diferenza dos procesos, non posúen un espazo de memoria propio, senón que o comparten co fío que os crea. A posibilidade de compartir o espazo de memoria brinda varias vantaxes e inconvenientes: é posible a comunicación implícita mediante variables, evitando a sobrecarga das funcións de paso de mensaxes, pero, por outro lado, pódense ocasionar problemas de sincronización ao acceder varios fíos de forma simultánea a algunha posición de memoria. É por isto que existen mecanismos para controlar os accesos concorrentes e, dalgunha maneira, paliar os problemas que poden causar. Algúns destes mecanismos son os *mutex* (exclusión mutua) ou os semáforos.

Aínda que unha aplicación implementada exclusivamente con MPI pode aproveitar todo o hardware dos clústeres (empregando un proceso por núcleo), o uso dunha aproximación híbrida con procesos que creen fíos para explotar os núcleos dun mesmo nodo ten as seguintes

vantaxes:

- Redución do tempo de xerar e destruír os elementos paralelos de procesado, xa que o mecanismo de xeración e destrución de fíos adoita ser menos pesado que o dos procesos de MPI.
- Redución da sobrecarga de memoria, xa que os fíos poden acceder a estruturas de datos compartidas, mentres que os procesos de MPI requiren unha copia destas estruturas para cada proceso.
- Posibilidade de executar en modo *Simultaneous MultiThreading (SMT)*. O *SMT* é unha tecnoloxía implementada nas CPUs que busca achegar unha mellora do rendemento no uso de fíos. Baséase na simulación de dous fíos lóxicos nun só núcleo físico do procesador, de forma que as instruccións dun dos fíos se intercalan coas do outro, aproveitando os ciclos de CPU que estarían libres. Esta técnica é máis coñecida comunmente como *Hyperthreading*.

3.2 Traballo relacionado

O emprego de información mutua é moi común no campo da selección de características, existindo moitas implementacións secuenciais baseadas nos métodos explicados no capítulo anterior. Por exemplo, en [12] introdúcense dous métodos similares a mRMR, pero estendendo relevancia e redundancia a relevancia condicional e redundancia condicional, co que se consegue mellorar a precisión. En [13] proponse unha variación de CondMI que reduce o seu alto custo computacional ao aplicar varias suposicións, e introduce unha compoñente para diferenciar entre características redundantes e irrelevantes. Por outra parte, en [14] inténtase adaptar a selección de características para que poida executarse en dispositivos de baixas capacidades, cada vez máis comúns co auxe do uso de dispositivos *wearables*. Despois de investigar sobre cada método de selección de características en particular, atopámonos con artigos que tentan unificar e realizar unha comparación de varios. Este é o caso de [1], que en concreto trata de atopar a relación entre métodos baseados en información mutua ao propoñer a selección de características como un problema de probabilidade condicional.

Por outra parte, a optimización destes métodos tamén aparece repetidas veces na literatura. Por exemplo, a versión rápida de CMIM é descrita en [9]. Na procura da mellora de rendemento, a computación de altas prestacións xa se utilizou para acelerar métodos de selección de características. Algúns traballos adaptan o deseño dun método concreto para usarse con MPI e/ou tecnoloxías multifío, isto é, a mesma aproximación que a utilizada neste traballo. Este é o caso de [15], que toma como base o método mRMR; [16], que propón un método *wrapper* baseado en algoritmos xenéticos paralelos; ou [17], que introduce un novo

método "online" (i.e. non se conta con todos os datos antes de aplicar a selección de características, senón que se van descubrindo pouco a pouco) e a súa versión asíncrona. Por outra parte, tamén existen implementacións paralelas usando ferramentas de *big data* como Spark ou Hadoop, e que, a pesar de contar coa posibilidade de executarse nun clúster, as súas optimizacións están máis dirixidas á execución en entornos na nube. Para estes entornos atopámonos por exemplo con [18], que aplica o paradigma *MapReduce* ao método denominado "*Relief*". Tamén se fai uso deste paradigma en [19], aproveitando o potencial do mesmo en combinación coa computación evolutiva; e en [20], para adaptar a selección de características con aproximación positiva (*FSPA*) a datasets de gran tamaño. En [21] preséntanse optimizacións con tecnoloxías multifío e con Spark para varios métodos da ferramenta Weka [22]. A maiores, tamén existen implementacións para outros tipos de hardware, como GPUs e FPGAs. Por exemplo, en [23] proporciónase unha optimización para o método JMI que utiliza a tecnoloxía CUDA, presente nas GPUs de Nvidia. Outras aproximacións que tamén fan uso de CUDA son [24], cunha optimización para *SVD* (*Singular Value Decomposition*, unha técnica de preprocesado para conxuntos con datos difusos), e [25], que mellora un método para a eliminación de ruído en imaxes cerebrais obtidas por resonancia magnética. Por outra parte, en [26] utilízase OpenCL (outro entorno para computación en GPUs, pero non privativo) para a distribución da carga de traballo de algoritmos evolutivos para a selección de características de electroencefalogramas.

Cabe destacar que non se atopou ningunha investigación que busque unha implementación paralela para varios métodos de selección de características ou proporcione unha libraría de métodos paralelos a potenciais usuarios, o que remarca o carácter innovador deste traballo.

3.3 Distribución de datos e de traballo

3.3.1 Descomposición de dominio

Para poder facer uso da computación paralela descrita anteriormente, é necesario adoptar unha estratexia de descomposición do problema inicial. Existen técnicas que se centran en repartir as diferentes fases do cómputo, de forma que cada elemento de procesado traballa cos datos que recibe do anterior, e delégaos no seguinte unha vez aplicado o seu procedemento. Isto coñécese como *pipeline*. Porén, o programa que estamos a tratar non ten diferentes fases de cálculo, senon que o mesmo tipo de cálculo repítese para diferentes datos. Neste caso a mellor aproximación é a que se coñece como descomposición de dominio.

A descomposición de dominio baséase en que todos os elementos de procesado realizan o mesmo tipo de cálculos, pero cada un sobre datos diferentes. A vantaxe con respecto a unha execución secuencial é que os datos de entrada se reparten entre ditos elementos para que a carga de cada un se vexa reducida. En concreto, a técnica escollida neste caso é a

descomposición de dominio por bloques, é dicir, os datos cos que traballará cada elemento de procesado están situados de forma contigua na estrutura inicial.

Dado un número *n* de elementos a repartir, e un número *p* de procesos, pódese calcular o tamaño do bloque (i.e. número de elementos por proceso) como $m = \lceil \frac{n}{p} \rceil$. Isto pode dar lugar a dúas situacións:

1. $n \mod p = 0$. O reparto é equitativo e todos os procesos traballarán con bloques de *m* elementos. Este é o caso óptimo, pero depende do dataset de entrada e das características do sistema no que se execute, polo que non temos certeza de que se vaia producir.

2. $n \mod p \neq 0$. O reparto non é equitativo, todos os procesos no intervalo $[0, p-2]$ traballarán con bloques de *m* elementos, pero o último $(p-1)$ recibirá $m' = n - m * (p-1), m' \in [1, m-1]$.

Como se viu, esta implementación básica causa que na maioría dos casos un proceso reciba menos datos que os demais, o que adoita que a súa carga de traballo tamén sexa menor e non se aproveite toda a súa capacidade. Polo tanto, foi necesario aplicar unha técnica un pouco máis avanzada para evitar isto. Entón, calcúlase $m = \lfloor \frac{n}{p} \rfloor$ e $r = n \mod p$. Isto quere dicir que, se o tamaño do bloque fose *m*, sobrarían *r* elementos. Eses elementos extra repártense engadindo un máis a *r* dos *p* procesos. Así, conséguese minimizar a diferenza entre os procesos a un máximo dun só elemento, de forma que a carga está mellor equilibrada entre eles.

3.3.2 Adaptación da descomposición de dominio á selección de características

Os algoritmos de selección de características cos que se está a traballar seguen unha estrutura similar, onde o único que cambia é o cálculo da puntuación de cada característica. Na Figura 3.7 móstrase o pseudocódigo xenérico deste proceso.

O programa comeza determinando a primeira característica a seleccionar, o que se fai sempre calculando a puntuación de todas as características como a información mutua coa clase (liñas 2 a 4) e seleccionando a de maior puntuación, i.e. a máis relevante (liñas 5 e 6). A continuación, para cada característica que resta por seleccionar (bucle da liña 9), búscase a seguinte mellor. Para isto, calcúlase a puntuación de todas as características que non foron seleccionadas ata o momento (liñas 11 a 15), e escóllese a mellor, que pasa a formar parte do conxunto de características seleccionadas (liñas 17 e 18). Cabe destacar que o cálculo da puntuación depende do método específico que esteamos a utilizar.

Como xa se comentou con anterioridade, existen dous niveis de paralelismo que se poden aproveitar para optimizar a execución de certos programas.

```
Entrada: Matriz de características M e número de características a seleccionar s
Saída  : Vector de índices selectedIndexes e vector de puntuacións
         selectedScores
A primeira característica é a que garda maior información mutua coa clase
foreach feature f_i ∈ M do
    classMI[i] ← calcMutualInformation(f_i, classColumn)
end
selectedIndexes[0] ← argmax(classMI)
selectedScores[0] ← max(classMI)

Agora buscamos as seguintes s − 1 mellores características
for i ← 1 to (s-1) do
    Calculamos unha puntuación para cada característica restante
    foreach feature f_j ∈ M do
        if feature j not yet selected then
            featureScores[j] ← ... /* Depende do algoritmo */
        end
    end
    Escollemos a característica con maior puntuación
    selectedIndexes[i] ← argmax(featureScores)
    selectedScores[i] ← maxfeatureScores
end
```

Figura 3.7: Pseudocódigo secuencial xenérico

- **Paralelismo explícito**. Este nivel utiliza procesos, sincronizados mediante paso de mensaxes. Como se explicou anteriormente, neste punto cada proceso está a traballar sobre unha parte dos datos totais, que é a que ten almacenada en memoria. A idea a aplicar neste caso é a seguinte:

    1. Para cada característica, cada proceso realiza os cálculos necesarios e atopa a que ten maior puntuación dentro do seu subconxunto local.
    2. Posteriormente, realízase unha operación de comunicación para atopar a maior puntuación global. Para isto, emprégase a función *MPI_Allreduce* co operador *MPI_MAXLOC*, onde os datos que proporciona cada proceso é un par co seu identificador de proceso e a puntuación máxima obtida. Deste xeito, cada un coñece cal é a máxima puntuación e que proceso a atopou.
    3. Por último, o proceso que atopou a mellor característica envía os datos da mesma a todos os demais, xa que moitos dos algoritmos utilizan as características xa seleccionadas para calcular a puntuación das seguintes. Para isto, utilízanse dúas operacións de *MPI_Bcast*: unha para enviar os datos da característica, e outra para

compartir o seu índice global (que é o que se obtén na saída do programa). Posteriormente, se aínda quedan características por seleccionar, voltamos ao paso 1.

- **Paralelismo implícito**. Neste nivel, os elementos de procesado son os fíos. Un proceso conta, incialmente, cun único fío, pero pode crear máis para distribuír o traballo. Neste caso, os fíos utilízanse para distribuír o cálculo das puntuacións das características que corresponden a un mesmo proceso. A idea é similar á proposta para o caso do paralelismo explícito: cada fío calcula as puntuacións das características que lle corresponden e selecciona a maior entre elas. Unha vez que rematan todos, escóllese o máximo de todos os fíos. Porén, neste caso non son necesarias as comunicacións, xa que ao compartir o espazo de memoria, o fío mestre (o que crea aos fíos que realizan o cómputo), proporciona a eses fíos unha variable na que almacenar os resultados.

3.4 Deseño software

Para a implementación do sistema desenvolvido neste Traballo Fin de Grao tomáronse como base as librarías MIToolbox e FEAST. MIToolbox contén as ferramentas para o cálculo das métricas de información mutua tratadas previamente. FEAST fai uso das funcións de MIToolbox para implementar os métodos de selección de características explicados no capítulo anterior, tanto nas súas versións estándar como nas *weighted* (ver Sección 2.4.9). Como é importante coñecer o impacto das melloras de rendemento que achegan as novas implementacións, Parallel-FST permitirá a execución dos métodos orixinais (i.e. as versións secuenciais) para poder obter uns tempos de referencia. Ademais, para evitar a repetición de código, o cálculo das métricas de información realizarase coas funcións de MIToolbox sempre que sexa posible.

Estas librarías están escritas en C polo que, para poder utilizalas sen ter que portalas a outra linguaxe, o proxecto debe facer uso de C/C++. Cabe destacar que se decidiu implementar o proxecto en C++ para manter a compatibilidade con MIToolbox, FEAST e MPI (C foi unha das linguaxes para as que se deseñaron as interfaces da súa especificación orixinal), e ademais poder facer uso da orientación a obxectos nas partes novas e menos críticas do programa, que son as que se listan a continuación:

- **Argumentos de liña de comandos** (Figura 3.8). O control do funcionamento do programa realízase a través da liña de comandos. A clase *Options* encárgase de realizar a análise dos argumentos e, cando non hai erros, permite acceder aos valores que se precisen. Ademais, creouse o enumerado *AlgorithmType* para indentificar o tipo de algoritmo que se debe aplicar (tentando evitar o uso de *strings*), e a clase *AlgorithmSelector* como conversor entre os valores do enumerado e as súas correspondentes cadeas de texto.

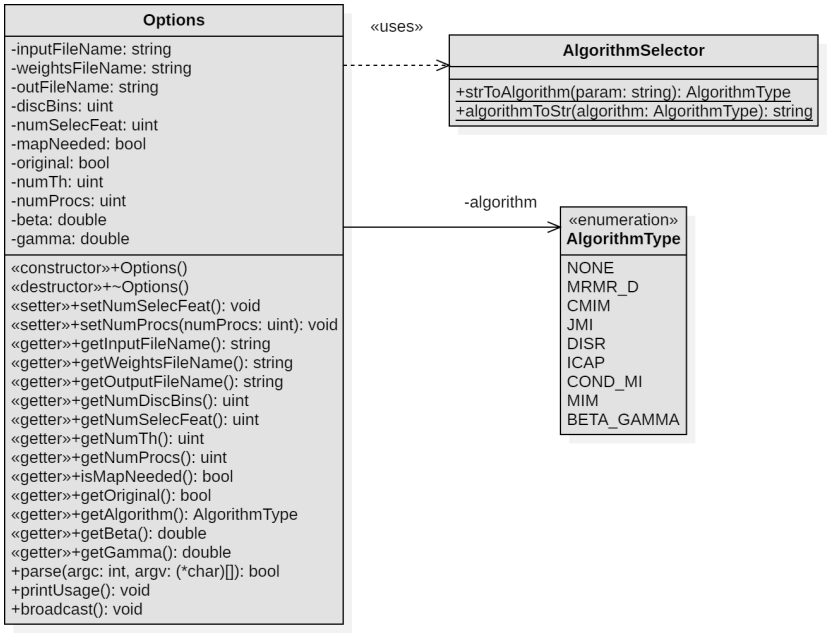


Figura 3.8: Diagrama das clases que xestionan os argumentos de liña de comandos

- **Lectura de datos** (Figura 3.9). Unha vez se sabe que os argumentos da liña de comandos son correctos, o seguinte paso é cargar os datos cos que vai traballar o algoritmo. Por unha parte temos o dataset, o que os algoritmos esperan como unha matriz. Para isto, utilízanse as clases *FileParser* e *InputMat*. *InputMat* representa e almacena os datos da matriz das características, e implementa as operacións de transformación dos datos como a discretización (descrita na Sección 2.3) e a compresión de rango (que se explicará na Sección 3.5.2). Por outra parte, *FileParser* encárgase de crear o obxecto da clase *InputMat* a partir do nome do arquivo no que está almacenado. Debido a que se consideran varios formatos diferentes de entrada, aplicouse un patrón estratexia para que cada formato se xestione nunha clase propia. Isto tamén permite engadir novos formatos dunha forma fácil e rápida. Como se explicou anteriormente, algúns algoritmos contan coa versión *weighted*, para o que é necesario un vector de pesos. A clase *WeightVector* cumpre coa función de representar esta información, ademais de cargala dende disco.

- **Algoritmo** (Figura 3.10). A clase *FeatureSelectionAlgorithm* representa ao algoritmo, cos seus datos e opcións para a execución. Este grupo de clases de todos os algoritmos deseñouse cun patrón estratexia, xa que tanto os datos que reciben como os resultados que producen todos os algoritmos seguen un mesmo formato. Ademais, deste xeito simplifícase a inclusión de novos algoritmos á libraría. Por outra parte, cada algoritmo pódese executar de varias maneiras: *weighted* ou non, secuencial ou paralelo. Para

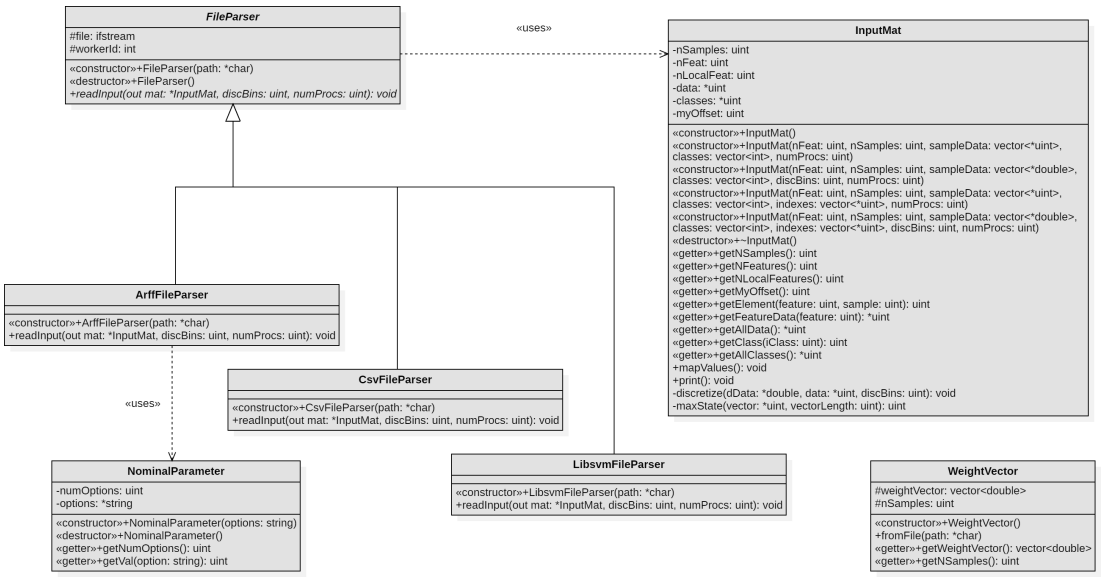


Figura 3.9: Diagrama das clases que xestionan a lectura de datos

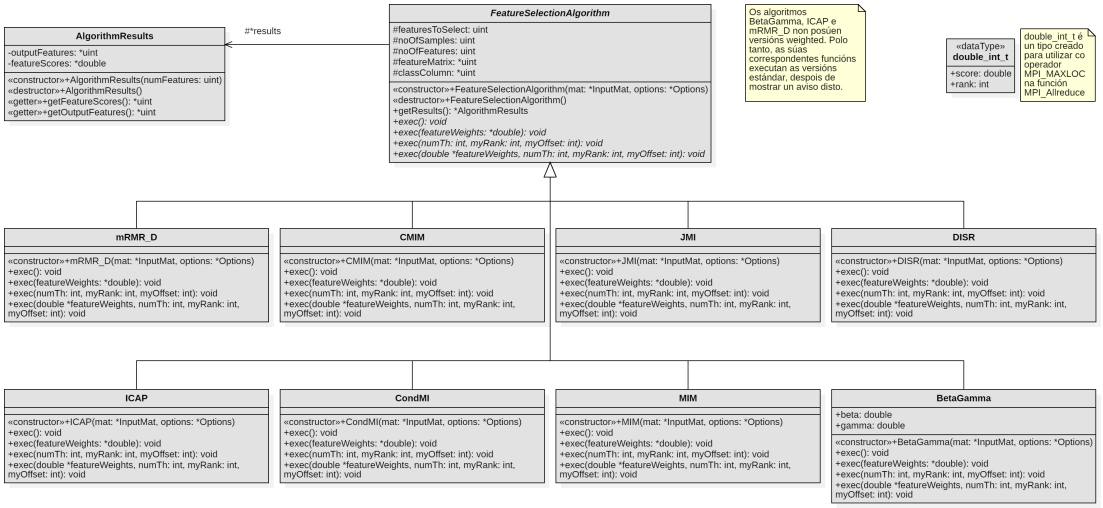


Figura 3.10: Diagrama de clases dos diferentes algoritmos

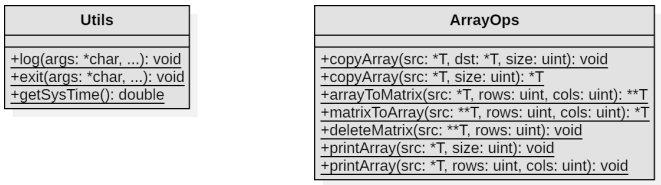


Figura 3.11: Clases e funcións auxiliares

permitir esta variabilidade, utilizouse a sobrecarga da orientación a obxectos co método "*exec*". Por último, engadiuse a clase *AlgorithmResults* para encapsular os resultados da execución do algoritmo. Cabe destacar que, con intención de preservar os nomes das funcións utilizadas da libraría orixinal, a clase que representa ao método "*mRMR*" recibe o nome de "*mRMR_D*".

- **Funcións auxiliares** (Figura 3.11). Por último, a clase *Utils* achega varias funcións estáticas de utilidade ao longo do programa. *ArrayOps*, pola súa parte, é unha microlibraría de funcións sobre arrays de tipos xenéricos que, aínda que se modelan como unha clase, foron implementadas como funcións soltas nun arquivo de código.

Como o obxectivo do proxecto é aproveitar as arquitecturas dos sistemas de computación distribuída, é necesario repartir datos e cómputo. Neste caso, as clases que dalgunha maneira precisan comunicacións entre procesos son as seguintes:

- *Options*: todos os procesos necesitan a información do número de características a seleccionar, número de procesos, etc. O método "*Options.broadcast*" (ver Figura 3.8) cumpre coa función de compartir estes datos, unha vez que un dos procesos analizou os argumentos de entrada. A compartición do obxecto, como ten que conter os mesmos valores en todos os procesos, baséase na operación *MPI_Bcast*, coa que se vai enviando cada atributo un a un. Cabe destacar que, no caso de atributos de lonxitude variable (como *strings*), son necesarias dúas comunicacións: a primeira, para coñecer a lonxitude dos datos a recibir, e a segunda, para os propios datos.
- *InputMat* (Figura 3.9): hai datasets que non caben na memoria dun nodo, polo que cada proceso só mantén a parte que necesita para realizar os cálculos. A distribución da matriz realízase ao crear o obxecto desta clase, de forma que os datos que almacenará este obxecto en cada proceso serán diferentes. Para esta distribución implementáronse dúas versións, en función do formato do dataset:
  - Cando o dataset está en formato denso, o proceso raíz ten estes datos almacenados como unha matriz, polo que coñece toda a información. Polo tanto, o primeiro paso é compartir o número de mostras e características con todos os procesos (*MPI_Bcast*), para que cada un coñeza a cantidade de datos a recibir. Posteriormente, con outra operación *MPI_Bcast*, envíase a columna de clases (i.e. un array que contén unicamente as clases de cada mostra). Por último, repártese a matriz cunha operación de *MPI_Scatter*. Deste xeito, cada proceso conta cos datos iniciais cos que vai traballar.
  - Se o dataset está en formato disperso, aplícase a "expansión selectiva" (ver Figura 3.13) que se explica en detalle na Sección 3.5.1. Dende o punto de vista da

implementación, e ao igual que no caso anterior, o primeiro paso é que todos os procesos coñezan a cantidade de datos a recibir, polo que se utiliza unha operación *MPI_Bcast* para compartir o número de mostras e características. Posteriormente, unha a unha, envíase cada mostra con *MPI_Bcast*. Por último, con outro *MPI_Bcast* envíase a columna de clases. Agora, cada proceso contén os mesmos datos, e é responsabilidade de cada un escoller os valores para as características que lle corresponden para construír o seu subconxunto local.

- *FeatureSelectionAlgorithm* (Figura 3.10): representa o estado do algoritmo, e créase a partir de *Options* e *InputMat*. A distribución desta clase está na execución do algoritmo (ver ítem "Paralelismo explícito" na Sección 3.3.2), xa que o obxecto *FeatureSelectionAlgorithm* de cada proceso traballa sobre os seus propios datos realizando as comunicacións oportunas cos demais.

Na Figura 3.12 amósase un esquema da interacción entre procesos e os obxectos das clases explicadas anteriormente. As partes que corresponden á lectura distribuída e á compresión de rango non se mostran ao completo, xa que se explican máis adiante.

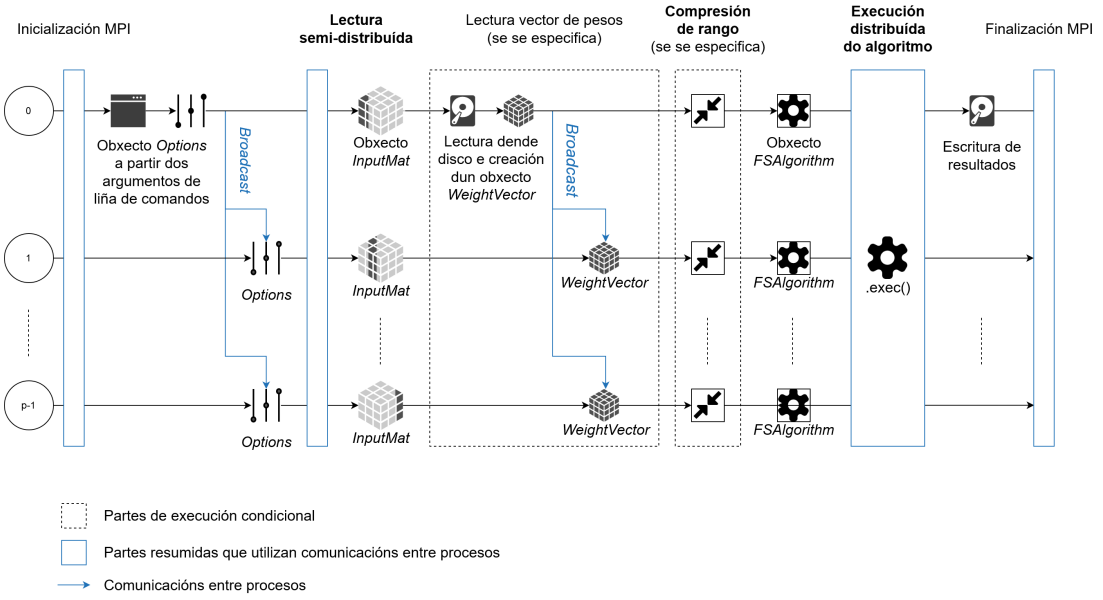


Figura 3.12: Esquema xeral do programa

3.5 Técnicas de implementación para mellorar o rendemento e a usabilidade

Parallel-FST implementouse tendo sempre en mente o obxectivo de obter o mellor rendemento posible en termos de tempo de execución, mantendo os mesmos resultados numéricos ca libraría secuencial FEAST na que se basea. Nesta sección explícanse algunhas técnicas que se incluíron de cara a obter un mellor rendemento máis aló da paralelización utilizando MPI e fíos. Ademais, tamén se incluíu algunha mellora de cara a facilitar o emprego da ferramenta polos seus futuros usuarios.

3.5.1 Lectura semi-distribuída dos datos

Na actualidade existen conxuntos de datos de gran tamaño aos que se lles pretende aplicar a selección de características. O almacenamento destes conxuntos de datos podería requirir decenas ou centos de GBs, superando así o tamaño da memoria dos nodos de moitos clústeres. Isto significa que non é posible aplicar unha aproximación paralela na que un único proceso (chamado raíz) realice a lectura do conxunto de datos completo ao inicio da execución, o garde na memoria do seu nodo, e logo envíe aos demais procesos as características coas que vai traballar cada un.

Para solventar este problema, nun primeiro lugar considerouse unha lectura distribuída utilizando as funcións da libraría MPI. Isto permitiría que cada proceso lese o ficheiro dende un *offset* determinado. Porén, o programa adoita traballar con formatos de entrada nos que as filas representan mostras (unha mostra por liña, como no exemplo da Figura 2.1). Polo tanto, como as características están representadas por columnas, cada proceso tería que acceder a todas as liñas do ficheiro e procesalas para extraer as características que lle interesan, o que sería pouco eficiente.

Outra opción sería que o proceso raíz fose lendo e enviando a parte correspondente a cada un dos outros procesos. Deste xeito o proceso raíz nunca tería que almacenar o conxunto completo na súa memoria local, pero esta solución non permite aproveitar a arquitectura distribuída do sistema para acelerar este proceso.

Finalmente, a lectura implementada ideouse ao estudar o formato no que se almacenan os datos. Algúns datasets como, por exemplo, os utilizados en recuperación de información ou procesamento de linguaxe natural, almacenan unha gran cantidade de ceros. Estes datasets soen almacenarse nun formato no que se suprimen os valores que son cero para así aforrar espacio de disco. Estes formatos coñécense como dispersos (ou *sparse*, en inglés), pero, polo xeral, os algoritmos que os utilizan precisan matrices densas, polo que é necesario convertilos. Na Figura 3.13 achégase un esquema do proceso seguido para a lectura dos datasets que se gardan en formato disperso (normalmente os máis grandes e custosos de analizar): o proceso

raíz carga os datos do conxunto completo en formato disperso, que cabe en memoria, e envíao a cada un dos outros procesos para que expanda só a parte que lle corresponde e despois realice os cálculos coa mesma. Cómpre destacar que a representación das matrices expandidas en memoria faise coas características como filas e as mostras como columnas (ao revés que nos formatos dos arquivos de texto), xa que así se accede de forma máis eficiente ao sistema de memoria.

A lectura semi-distribuída pode ocasionar un problema cando hai moitos procesos nun mesmo nodo, xa que todos usan o mesmo módulo de memoria. Como cada un expande a parte do dataset que lle corresponde, é posible que a memoria dun nodo non sexa suficiente para almacenar tantos fragmentos expandidos. Nese caso a mellor aproximación consistirá en empregar varios fíos por proceso, xa que comparten datos.

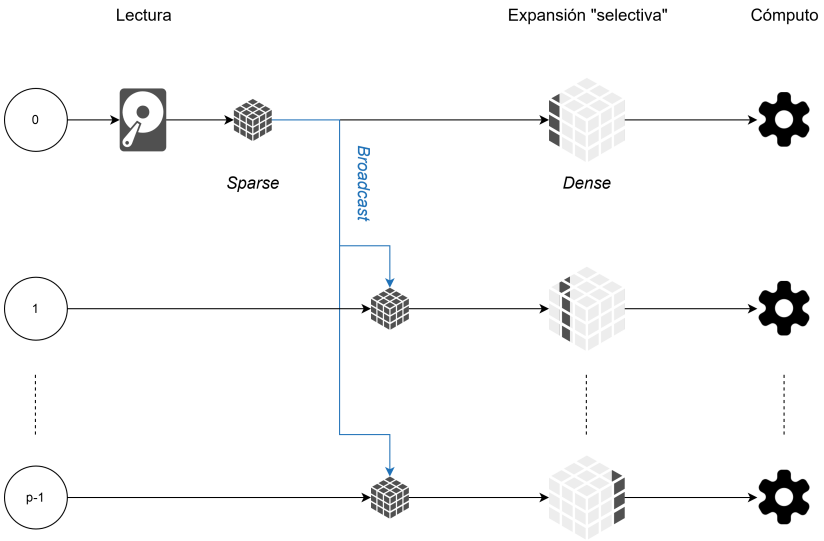


Figura 3.13: Esquema da lectura semi-distribuída

3.5.2 Compresión de rango

Tras unhas primeiras probas de rendemento, xurdiu o problema de que certos datasets de pequeno tamaño producían erros de esgotamento de memoria. Nestes conxuntos de datos sempre había algunha característica cuns valores que se atopaban nun rango moi extenso, pero o número de valores diferentes era moito menor que os valores do rango. Estes tipo de datasets, mesmo nos casos nos que non esgotaban a memoria, causaban que a execución fose moito máis lenta do normal.

Unha análise en profundidade dos algoritmos desvelou que o uso esaxerado de memoria se producía durante o cálculo da información mutua entre características. Este proceso baséase na creación dun histograma, é dicir, cóntase o número de ocorrencias de cada par de

Característica	Valores				
f1	0	2	2	0	1
f2	1	3	3	2	1

⟶

f1 \ f2	**0**	**1**	**2**	**3**
0	0	1	1	0
1	0	1	0	0
2	0	0	0	2

Figura 3.14: Matriz de ocorrencias cando o número de valores diferentes é igual que o número de valores posibles. Créase unha matriz 3x4

Caract.	Valores				
f3	0	1000	1000	0	1000
f4	1000	2000	2000	1000	1000

⟶

f3 \ f4	**0**	**...**	**1000**	**...**	**2000**
0	0	...	2	...	0
...	...	...	...	...	...
1000	0	...	1	...	2

Figura 3.15: Matriz de ocorrencias cando o número de valores diferentes é moito menor que o número de valores posibles. Créase unha matriz 1001x2001

valores, polo que se crea unha matriz como o produto cartesiano dos posibles valores de ambas características. Nas Figuras 3.14 e 3.15 amósase dita matriz para un caso normal, e para o que produce un alto consumo de memoria, respectivamente.

Como se pode observar nos exemplos, a presenza de valores moi altos para unha característica fai que o histograma que se crea sexa dun gran tamaño. Por unha parte, isto ocasiona un maior consumo de memoria, o que nalgúns casos pode causar o erro comentado previamente. Cando o erro non se dá, isto produce que o cómputo sexa máis lento, xa que se itera sobre unha gran cantidade de filas e columnas vacías (todos os valores son cero) que non inflúen no resultado.

Aínda que isto poida parecer un problema pouco común, non é tan raro atoparse cun caso como o seguinte: o cálculo entre as características das frecuencias mínima e máxima de CPUs, con valores en kHz en rangos [0, 700000] e [500000, 4000000]. Supoñendo que os contadores son enteiros de 4 bytes, a matriz necesaria ocuparía sobre 10TB. Porén, esta matriz podería reducirse moito, xa que é moi probable que as frecuencias estean aproximadas á orde de centenas ou miles, causando que na matriz aparezan filas e columnas cheas de ceros que non cumpren ningunha función.

A solución proposta consiste nun "renomeado" dos valores das características. Xa que o cálculo da información mutua só utiliza o número de ocorrencias, e non os valores en si, isto non supón cambios para os resultados dos cálculos. Outra vantaxe desta solución é que se pode implementar con complexidade O(n), polo que non supón un gran aumento no tempo de cómputo total. De feito, en xeral, ao reducir o tamaño das matrices, o tempo que se aforra nas iteracións das mesmas compensa sobradamente o tempo utilizado para este procedemento,

Entrada: Unha matriz M de tamaño $F \times S$
Saída : A matriz M cos valores actualizados

```
1  foreach fila f ∈ M do
2      maxState ← max(f)
3      featMap ← zeros(maxState)
4      mapCounter ← 1
5      foreach valor s_i ∈ f do
6          s ← f[i]
7          if featMap[s] = 0 then
8              /* Primeira vez que aparece s */
9              featMap[s] ← mapCounter
10             mapCounter ← mapCounter + 1
11         end
12         f[i] ← featMap[s] - 1
13     end
14 end
```

Figura 3.16: Pseudocódigo da compresión de rango

como se demostrará experimentalmente na Sección 4.2.

O pseudocódigo da solución preséntase na Figura 3.16. Como se pode observar, o tratamento de cada característica (fila) é independente. A idea consiste en atopar un novo valor que substitúa cada valor diferente da característica, asegurando que o rango que poden tomar estes valores sexa mínimo. Isto conséguese coas variables *featMap* e *mapCounter*: a primeira é un vector que se utiliza para establecer a relación entre valores iniciais e novos (*novo = vector[inicial]*); mentres a segunda é o contador que leva o número de valores diferentes ata o momento, que é o que ademais permite obter un novo nome cando aparece un valor non visto ata o momento. Nas Figuras 3.17 e 3.18 móstrase como afectaría este procedemento aos exemplos propostos anteriormente nas Figuras 3.14 e 3.15, respectivamente.

Dous aspectos que en principio poderían crear problemas son os seguintes:

- **Necesidade de desfacer este proceso**. É dicir, volver a substituír os valores novos polos orixinais. Este proceso non é necesario, xa que os valores concretos non se utilizan en ningún momento no programa e os resultados que se producen son só os índices das características a seleccionar e as súas puntuacións.

- **Compatibilidade coa lectura semi-distribuída**. Pódese pensar que ao distribuír as características entre os distintos procesos, poderíanse dar problemas de conflitos entre os novos valores. Porén, como o que se reparte son as características ao completo, este non é un problema, xa que o procedemento é intra-característica. A maiores, isto

Característica	Valores				
f1'	0	1	1	0	2
f2'	0	1	1	2	0

⟶

f1' \ f2'	**0**	**1**	**2**
0	1	0	1
1	0	2	0
2	1	0	0

Figura 3.17: Compresión de rango, créase unha matriz 3x3 en lugar dunha 3x4. Non hai ningunha fila ou columna na que todos os valores sexan cero

Característica	Valores				
f3'	0	1	1	0	1
f4'	0	1	1	0	0

⟶

f3' \ f4'	**0**	**1**
0	2	0
1	1	2

Figura 3.18: Compresión de rango, créase unha matriz 2x2 en lugar dunha 1001x2001. Non hai ningunha fila ou columna na que todos os valores sexan cero

permite que esta técnica tamén se realice de forma paralela, repartindo o cómputo entre os procesos dispoñibles.

Por último, mencionar que durante o desenvolvemento de Parallel-FST a esta solución déuselle o nome de "mapeo", pero cambiouse posteriormente xa que non era totalmente axeitado. Por este motivo nalgúns diagramas do deseño e no código esta operación aínda mantén o nome de *mapeo* ou *mapValues*.

3.5.3 Automatización da compilación

Como se comentou anteriormente, este proxecto fai uso doutras librarías. Por iso, para levar unha boa xestión do mesmo, utilizouse a ferramenta CMake[5] para o proceso de compilación. CMake permite especificar os requisitos para a compilación do proxecto con independencia da plataforma na que se utilice. Así, ademais de evitar ter que crear *Makefiles* con instruccións complexas, conséguese un proxecto portable a un maior número de sistemas. Grazas a isto conséguese mellorar a usabilidade do programa, xa que é accesible para un maior público, con independencia dos niveis de coñecementos informáticos e da plataforma na que se utilice.

[5] https://cmake.org/

Capítulo 4

Resultados experimentais

Neste capítulo describirase o entorno e os datos utilizados nas probas, así como o proceso de selección da mellor configuración para as medidas finais. Ademais, presentaranse para cada algoritmo a mellora de rendemento obtida tras aplicar as técnicas de paralelización e optimización propostas, así como unha breve conclusión para cada un.

4.1 Entorno de probas

4.1.1 Sistema

As probas de rendemento realizáronse nun clúster do CITIC[1] denominado "Plutón"[2]. Este clúster é un sistema heteroxéneo orientado á computación de altas prestacións, e está composto de nodos conectados por unha rede de altas prestacións (como se describiu na Sección 3.1 do capítulo anterior). A Figura 4.1 amosa a estrutura do clúster.

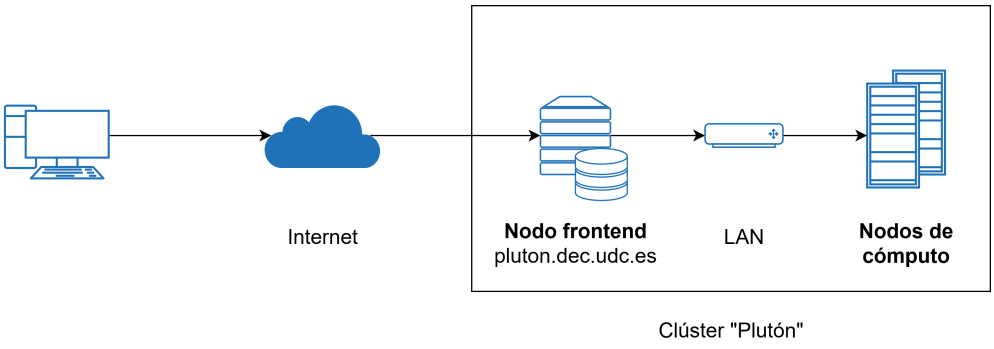


Figura 4.1: Estrutura xeral do clúster Plutón

En concreto, "Plutón" conta cun nodo *frontend*, que cumpre a función de punto de entrada único dende o exterior. Dende este nodo pódese editar e compilar o código, así como enviar

[1] https://www.citic-research.org/
[2] http://pluton.dec.udc.es/

traballos ao planificador, ou sistema de colas, que é o encargado de poñelos en execución nos nodos de cómputo. O *frontend* tamén realiza a función de servidor NAS (*Network Attached Storage*) no que se almacenan os ficheiros dos usuarios. Os nodos de cómputo poden acceder a estes datos de forma remota a través da rede local.

Por outra parte, os nodos de cómputo son os que proporcionan os recursos computacionais: CPU, memoria e aceleradoras (GPUs). Están organizados de forma lóxica por cabinas de nodos con recursos similares. Actualmente, conta con dúas cabinas, distribuídas da seguinte maneira:

- Cabina 0: 18 nodos de cómputo (compute-0-0 a compute-0-17), cun total de 288 núcleos, 1152GB de memoria e 21 aceleradoras.

- Cabina 1: 2 nodos de cómputo (compute-1-0 e compute-1-1), cun total de 48 núcleos e 256GB de memoria.

Neste Traballo Fin de Grao empregamos a Cabina 0, xa que ten unha maior cantidade de nodos e permítenos medir a escalabilidade de Parallel-FST. As especificacións hardware concretas dos nodos de cómputo desta cabina amósanse na Táboa 4.1. Cómpre destacar que cada nodo ten dous procesadores octa-core, co que se poden usar ata 16 núcleos por nodo (ata 32 fíos se explotamos o *Hyperthreading*).

	compute-0-{0-17}
CPU (modelo)	2 × Intel Xeon E5-2660 Sandy Bridge-EP (0-{0-16}) 2 × Intel Xeon E5-2650v2 Ivy Bridge-EP (0-17)
CPU (velocidade/turbo)	2.20GHz / 3.0GHz (0-{0-16}) 2.60GHz / 3.4GHz (0-17)
Núcleos por CPU	8
Fíos por núcleo	2
Núcleos/Fíos por nodo	16 / 32
Caché L1/L2/L3	32KB / 256KB / 20MB
Memoria RAM	64GB DDR3 1600Mhz
Discos	1 × HDD 1 TB SATA3 7.2K rpm
Redes	InfiniBand FDR e Gigabit Ethernet

Táboa 4.1: Especificacións dos nodos da Cabina 0

Na Figura 4.1 pódese observar que todos os nodos están interconectados por unha rede LAN. Esta rede é unha rede Gigabit Ethernet, cun ancho de banda máximo de 1Gbps. Ademais, como se amosa na táboa de especificacións, todos os nodos contan cunha interface de rede InfiniBand FDR. InfiniBand é unha rede de interconexión de altas prestacións, é dicir, cun alto ancho de banda e baixa latencia, que conta con varias especificacións. No caso deste clúster,

a especificación utilizada é FDR, o que permite anchos de banda de ata 56Gbps e latencias ao redor de 1-2μs.

Como se comentou previamente, o acceso aos nodos de cómputo non é directo, senón que está xestionado por un planificador, tamén coñecido como xestor de traballos ou sistema de colas. Este software encárgase de asignar os recursos do clúster aos diferentes usuarios segundo as súas necesidades. Os usuarios envían os traballos dende o nodo *frontend* indicando os recursos requeridos, o que lle permite ao planificador poñelos en execución cando o considere oportuno, sempre tentando satisfacer a os requerimentos da maior cantidade posible de usuarios. O sistema de colas instalado en "Plutón" é Slurm Workload Manager[3], na versión 19.05.2. A unidade de execución de Slurm é o traballo (*job*, en inglés), que pode ser de dous tipos: *batch* ou interactivo. Ao enviar un traballo ao planificador, Slurm acepta parámetros para establecer os recursos necesarios. A continuación lístanse os máis relevantes durante a realización das probas deste Traballo Fin de Grao:

- **Memoria**. A memoria que necesita o noso traballo. Pódese solicitar en total ou por núcleo. Hai que ter en conta que está limitada pola memoria RAM dos nodos que estamos a utilizar (64GB).
- **Número de nodos**. O número de nodos que queremos utilizar. Pódese solicitar en total, ou podemos especificar o número de procesos totais e o número de procesos que queremos executar en cada nodo, de modo que Slurm calcula os nodos totais.
- **Número de núcleos**. Indica o número de núcleos que precisa o traballo en execucións con paralelismo implícito. Pódese especificar en total, ou por proceso.
- **Exclusividade**. Esta opción impide que o sistema de colas poña en execución traballos doutros usuarios nos nodos nos que se está a executar o noso, asignando todos os núcleos e toda a memoria do nodo. É moi útil para probas de rendemento, xa que se evitan as interferencias que outros traballos poidan introducir nas nosas medidas.
- **Tempo de execución**. Debemos indicar de xeito obrigatorio o tempo de execución estimado do noso traballo. É importante tentar realizar unha boa estimación, xa que así contribuímos a un funcionamento máis eficiente do planificador. Por outra parte, para asegurar a calidade do servizo e un bo reparto do hardware entre todos os usuarios do clúster, existe un límite máximo de 72h para todos os traballos.
- **Recursos especiais**. Aínda que non se utilizaron neste traballo, este é un parámetro que cómpre mencionar, xa que é o que permite solicitar aceleradoras, como as GPUs ou as unidades Xeon Phi.

[3] https://slurm.schedmd.com/

Non se especifica a sintaxe dos parámetros, xa que isto é algo que se pode atopar na documentación de Slurm[4].

Outra das tarefas principais do nodo *frontend*, como se comentou anteriormente, é a compilación dos programas. Este clúster ten instalada a ferramenta Lmod [5] para administrar a maioría do software co obxectivo de poder soportar distintas versións dun mesmo paquete, libraría ou aplicación. Isto simplifica os cambios entre versións, xa que xestiona automaticamente as variables de entorno e non é necesario realizar modificacións no *path*. En concreto, os módulos e librarías utilizados para a compilación do sistema desenvolvido foron os seguintes. Cabe destacar que os tres primeiros, ao estar instalados no sistema, puidéronse cargar con Lmod.

- **CC**. Este módulo achega un compilador de C/C++, ademais de librarías estándar necesarias, polo que debe estar cargado tanto para a compilación como para a execución. En particular, utilizouse a implementación de GNU (gcc)[6], na súa versión 8.3.0.
- **MPI**. Esta libraría, descrita na Sección 3.1.1, especifica as funcións utilizadas para a comunicación por paso de mensaxes, e é necesaria para a compilación e para a execución. En concreto utilizouse a implementación de OpenMPI[7], na súa versión 3.1.4.
- **CMake**. É o programa utilizado para a automatización da compilación do proxecto, tal como se comentou na Sección 3.5.3. En concreto, utilizouse a versión 3.15.4.
- **MIToolbox e FEAST**. Estas dúas librarías, descritas nos capítulos anteriores, implementáronse para o traballo descrito en [1], e están dispoñibles no repositorio público do autor[8]. MIToolbox contén funcións para o cálculo de métricas de cantidade de información, mentres que FEAST inclúe as implementacións secuenciais orixinais dos métodos de selección de características. Como non son librarías estándar, non é habitual que estean instaladas nos clústeres, polo que se incluíron no código do proxecto. Isto, ademais, permite que CMake xestione a súa compilación como librarías dinámicas, para que posteriormente sexan cargadas polo programa xeral e usadas na execución.

4.1.2 Datasets

As probas de rendemento miden as melloras que achegan as técnicas de paralelización e optimización propostas. Seleccionamos varios datasets con diferentes características de forma que temos un exemplo representativo das variantes que se dan habitualmente. Concre-

[4] https://slurm.schedmd.com/documentation.html
[5] https://lmod.readthedocs.io
[6] https://gcc.gnu.org/
[7] https://www.open-mpi.org/
[8] https://github.com/Craigacp

Dataset	*Features*	*Samples*	Clases	Tipo de dato	Tamaño (8B/dato)
Breast	24481	97	2	real	18.12 MB
ECML	27679	90	43	enteiro	19.01 MB
Epsilon	2000	400000	2	real	6.96 GB
RCV1	47236	20242	2	real	7.12 GB
News20	62061	15935	20	enteiro	7.37 GB
SVHN	3072	531131	10	enteiro	12.16 GB
E2006	4272227	16087	–	real	512.06 GB

Táboa 4.2: Datasets das probas

tamente, tentamos comprobar a escalabilidade da solución ante datasets que conteñen máis características que mostras e viceversa, e tamén para aqueles que conteñen datos de poucas e de moitas clases. A continuación achégase unha descrición de cada un dos conxuntos de datos utilizados. A maiores, na Táboa 4.2 pódese ver a relación de características (*features*), mostras (*samples*) e clases para cada un. Os cinco máis grandes (Epsilon, RCV1, News20, SVHN e E2006) atópanse dispoñibles na web da Universidade Tecnolóxica de Nanyang[9].

- **Breast** e **ECML**. Estes dous datasets de menor tamaño só se utilizaron para as probas de validación, é dicir, para asegurar que os resultados da implementación paralela coincidían exactamente cos da implementación orixinal. O primeiro contén datos para a detección de cancro de mama e o segundo empregouse nun "*discovery challenge*" da *European Conference on Machine Learning (ECML)*.

- **Epsilon** [27]. dataset artificial creado para o "*Pascal large scale learning challenge*"[10] de 2008. Os datos están normalizados característica a característica para que a media sexa cero e a varianza un, e posteriormente normalizados para que a lonxitude de cada mostra sexa un. Este é un conxunto biclase, con máis mostras que características.

- **RCV1** [28]. Arquivo de noticias categorizadas a man, publicado pola axencia Reuters[11] para fins de investigación. Este conxunto tamén é biclase, pero cun número de características superior ao de mostras.

- **News20** [29]. Colección de 20000 documentos de noticias aproximadamente, particionada de xeito máis ou menos uniforme sobre 20 tipos de noticias diferentes. News20 cobre o espazo de datasets con moitas clases, e máis características que mostras.

- **SVHN** [30]. Conxunto de placas de números de casa en formato imaxe, con resolución 32x32. As características expresan os valores para as canles RGB de cada píxel. Este

[9] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
[10] https://www.k4all.org/project/large-scale-learning-challenge/
[11] https://www.reuters.com/

conxunto tamén conta cun número alto de clases, pero contén moitas máis mostras que características.

- **E2006** [31]. Corpus de informes anuais extraídos do "*Form 10-K*". Estes son informes de empresas dos Estados Unidos publicados entre 1996 e 2006, que conteñen información histórica, organizativa e financieira das mesmas. En principio é un dataset para regresión, polo que non existe unha característica que cumpra o papel de clase, senón que é un valor numérico arbitrario. O interesante deste dataset é o seu gran tamaño, xa que supera notablemente a memoria da que dispón un só nodo.

Os datasets almacénanse en memoria como matrices de números. Porén, para poder compartilos e transferilos, mantendo a posibilidade de utilizalos por outros programas, é necesario almacenalos en disco como texto nun formato determinado. Como xa se mencionou anteriormente, estes formatos poden ser densos (se almacenan todos os valores do dataset) ou dispersos (se eliminan determinados valores para aforrar espazo). Por exemplo, en datos de linguaxe, as características representan todos os termos dunha linguaxe, e o valor na mostra indica se o termo aparece (1) ou non (0), orixinando unha gran cantidade de ceros. Para aforrar espazo en disco só se almacenan os datos diferentes de 0. Se unha determinada posición non ten datos enténdese que o seu valor é 0. Os datasets que se utilizaron para as probas non se atoparon todos no mesmo formato, xa que os máis grandes fan uso de formatos dispersos, e os máis pequenos, de formatos densos. A continuación achégase unha lista dos formatos concretos utilizados:

- **ARFF** (*Attribute-Relation File Format*). É un formato denso desenvolvido especificamente para o software de aprendizaxe máquina de Weka[12][22].
- **CSV** (*Comma Separated Values*). Este formato é amplamente utilizado e coñecido, e almacena os datos como unha matriz (polo tanto, é un formato denso), na que as filas están separadas por saltos de liña, e os valores de cada columna por comas.
- **LibSVM**. Este é o formato propio da ferramenta LibSVM[13][32]. É un formato disperso que suprime os ceros, para o que é necesario outorgar un índice ás características. Así, unha mostra que só tivese un valor distinto de cero, representaríase como

$$< clase/valor\ esperado > \quad < indiceN > \ : \ < valorN >$$

A maioría dos datasets utilizados conteñen datos continuos aos que lles foi necesario aplicar o proceso de discretización mediante *binning* explicado previamente na Sección 2.3.

[12] https://www.cs.waikato.ac.nz/ml/weka/
[13] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

4.2 Selección da configuración para un nodo

A implementación paralela da libraría permite ao usuario escoller diferentes configuracións de cara a obter o mellor rendemento en diferentes arquitecturas. Antes de comezar coas probas de escalabilidade, é necesario atopar a configuración que mellor rendemento obtén nun único nodo de "Plutón". Asumimos que a mellor configuración para un nodo será a mellor tamén cando se aumente o número de nodos. Como se viu ata agora, os parámetros que podemos variar son a relación entre o número de procesos e o de fíos, o uso ou non de *Hyperthreading* (descrito na Sección 3.1.2), e o uso ou non da compresión de rango descrita na Sección 3.5.2.

En canto á ratio de fíos por proceso, cabe lembrar que os nodos da Cabina 0 do clúster "Plutón" teñen dúas CPUs con oito núcleos en cada unha. É dicir, en total, nun nodo, contamos con 16 elementos de procesado. Para aproveitar as capacidades do sistema ao máximo debemos utilizar o maior número de elementos de procesado posibles. Polo tanto, neste caso, temos que $nProcesos * Fios\_por\_Proceso = 16$. A maiores, está o *Hyperthreading*, que nos permite utilizar dous fíos por núcleo, polo que poderiamos utilizar o dobre de fíos por proceso (i.e. $nProcesos * Fios\_por\_Proceso = 32$). Porén, estes fíos son lóxicos, polo que o número de elementos de procesado segue a ser 16.

Para estas probas utilizáronse varios conxuntos de datos pequenos, xa que estamos limitados pola memoria dun só nodo. En concreto, para esta tarefa utilizáronse Breast, ECML e Epsilon. Ademais, o número de características a seleccionar fixouse en 200, e para aqueles datasets con valores reais utilizouse a discretización con 128 *bins*. As combinacións que se probaron foron para 1, 2, 4, 8 e 16 procesos empregando o máximo número de fíos posible para cada un, con e sen *Hyperthreading*. En todos os casos chegouse á conclusión de que a mellor configuración para o nodo consiste en lanzar 2 procesos (i.e. un proceso por procesador), con 8 núcleos asignados a cada proceso, facendo uso do *Hyperthreading* (i.e. 16 fíos lóxicos por proceso).

Por outra parte, estas probas tamén se utilizaron para avaliar os beneficios da compresión de rango. Como se explicou no capítulo anterior, este é un proceso de complexidade baixa e moi rápido. A mellora depende dos datos que conteña o conxunto orixinal, polo que se escolleu un caso ilustrativo para demostrar dita mellora. Na Táboa 4.3 amósanse os tempos de execución do método CondMI na súa versión secuencial orixinal, con e sen aplicación da compresión de rango, para a selección de 200 características.

Como se pode observar na última columna da táboa, a aplicación deste procedemento presenta un tempo de execución moi baixo en relación á execución do propio método. Ademais, a mellora que se produce é moi significativa, polo que se aplicará a compresión de rango nas probas de rendemento. Cabe destacar que a nosa suxerencia é activar sempre a selección des-

Dataset	Execución sen CR (s)	Execución con CR (s)	Execución da CR (s)
Breast	392.8807	145.8067	0.0136
ECML	10178.3957	93.5077	0.0115
Epsilon	178383.3230	20828.2496	2.2124

Táboa 4.3: Tempos obtidos pola execución secuencial con e sen compresión de rango (CR) cando se seleccionan 200 características

ta operación, xa que no peor caso nin melloraría nin empeoraría o tempo de execución do método, e a sobrecarga que introduce é mínima.

4.3 Probas de escalabilidade

Nesta sección expóñense os resultados das probas de rendemento para varios nodos. Para cada método e dataset seleccionaranse 200 características. Esta cantidade de características é suficientemente grande como para ser capaces de estudar o impacto da eliminación de características do conxunto de candidatas, mentres que ao mesmo tempo non é excesivamente grande, o que non sería útil nun escenario real. En tódolos casos calculouse o tempo de execución orixinal secuencial como base das comparacións, e os das execucións paralelas para 1, 2, 4, 8 e 16 nodos, coa configuración por nodo explicada na sección anterior. Concretamente, a configuración que se utiliza é: 2 procesos por nodo, 8 núcleos por proceso con *Hyperthreading* (16 fíos lóxicos por proceso), discretización con 128 *bins* e uso da compresión de rango. Porén, foi necesario establecer varias excepcións, debido a que xurdiron varios problemas durante as probas finais:

- SVHN e E2006 son os dous conxuntos de datos máis grandes dos utilizados. Debido a que a lectura semi-distribuída (Sección 3.5.1) envía todos os datos en formato disperso a cada proceso, cando hai máis dun proceso por nodo, supérase a memoria do mesmo. Polo tanto, para as probas con estes conxuntos de datos utilizouse 1 proceso por nodo, con 16 núcleos por proceso e *Hyperthreading* (32 fíos lóxicos por proceso).
- O método CondMI conta cunha complexidade superior aos demais, polo que, para conxuntos de datos grandes, os tempos de execución superan o límite de 72h do clúster. Polo tanto, para CondMI, o número de características a seleccionar reduciuse a 50. Por outra parte, este método tamén utiliza moita memoria, polo que foi necesario utilizar a configuración de 1 proceso por nodo, con 16 núcleos por proceso e *Hyperthreading* para todos os datasets, a excepción de SVHN, para o que tamén foi necesario desactivar o *Hyperthreading*.

Os tempos medidos achéganse en forma de táboa, e tamén se inclúe un gráfico de barras

para mostrar a aceleración. Cómpre advertir que estas gráficas seguen unha escala logarítmica de base dous no eixo vertical.

Antes de expoñer os resultados de cada algoritmo, é necesario comprender varios conceptos, entre eles as métricas de rendemento que se calcularon e cal é o seu significado:

- **Elemento de procesado**. O elemento de procesado é a unidade de cómputo mínima que participa na execución. Cabe destacar que nos referimos a unidades físicas e non lóxicas, xa que, aínda que son as segundas as que permiten explotar as primeiras, as que realmente aportan a capacidade de cómputo son as físicas. Por exemplo, podemos executar dous fíos nun único núcleo, pero isto non aportaría o dobre de capacidade cómputo, xa que deben compartir o núcleo físico. Polo tanto, referímonos a elementos de procesado como os núcleos utilizados na execución. Neste caso en particular, o clúster utilizado conta con 16 en cada nodo, polo que os elementos de procesado p totais calcúlanse como $p = numNodos * numNucleos\_por\_nodo = numNodos * 16$.

- **Aceleración (ou *speedup*)**. A aceleración é a métrica que indica a relación entre o tempo de execución da ferramenta paralela e o da súa versión secuencial. Calcúlase, para p elementos de procesado, como:

$$speedup(p) = \frac{T_{secuencial}}{T_{paralelo}(p)} \tag{4.1}$$

  Onde $T_{secuencial}$ é o tempo total da execución secuencial, e $T_{paralelo}(p)$, o da execución paralela con p elementos de procesado. Idealmente toma valores entre cero e p. A continuación, explícanse os tres casos que se poden dar.

  - $speedup(p) \in (0, 1]$: A execución paralela non mellora o tempo da secuencial. Se é igual a un, a execución tarda o mesmo.
  - $speedup(p) \in (1, p\,]$: A execución paralela mellora o tempo da secuencial. A paralelización ideal conta cun $speedup = p$, é dicir, a execución secuencial repártese exactamente entre os elementos de procesado.
  - $speedup(p) > p$: A execución paralela supera a mellora teoricamente óptima. Este caso coñécese como *superlinealidade* e pódese dar por diversos factores, como que a execución paralela mellora a xestión da memoria, reducindo os fallos caché e consecuentemente os tempos de execución.

- **Eficiencia**. A eficiencia dá unha idea do comportamento do programa paralelo, independentemente do número de elementos de procesado nos que se execute. Toma valores entre cero e un (paralelización perfecta), aínda que pode tomar valores superiores a un

cando o *speedup* é superlineal. Tamén adoita expresarse en porcentaxe, multiplicando o valor por cen. Calcúlase, para p elementos de procesado, como:

$$eficiencia(p) = \frac{speedup(p)}{p} = \frac{T_{secuencial}}{p * T_{paralelo}(p)} \tag{4.2}$$

- **Escalabilidade**. Nun caso ideal, a eficiencia manteríase constante independentemente do número de elementos de procesado, pero na realidade non é así. A escalabilidade defínese como a capacidade de manter a eficiencia na resolución dun problema ao dispoñer de maiores recursos.

4.3.1 MIM

Este método, como se viu no Capítulo 2, posúe unha menor complexidade que os demais, xa que só calcula as puntuacións de cada característica unha vez para logo escoller as mellores. Na Táboa 4.4 amósanse os tempos de execución en diferentes nodos (N) para cada dataset. Como se pode observar, os tempos deixan de reducirse nas execucións con 8 ou 16 nodos. Isto débese a que, ao aumentar o número de nodos tamén aumentan as comunicacións necesarias, o que é significativo neste caso en particular, xa que a simplicidade do algoritmo permite que a execución se realice nun tempo moi pequeno. Recórdase que a configuración utiliza dous procesos por nodo, polo que o número de comunicacións multiplícase por dous en cada nodo utilizado. Cabe destacar que non se puideron obter os tempos das execucións en menos de 16 nodos para o dataset E2006 xa que, como se comenta na Sección 4.1.2, é un dataset de gran tamaño, polo que non cabe en memoria utilizando menos nodos.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	7.1173	1.5411	1.4360	0.9652	1.4202	1.6945
RCV1	10.1047	1.5514	1.0920	0.7941	0.7067	0.9051
News20	10.3263	1.5910	1.0393	0.7704	0.7482	0.8795
SVHN	13.5336	4.6332	3.3744	2.8326	1.5710	5.2889
E2006	–	–	–	–	–	17.0533

Táboa 4.4: Tempos do algoritmo MIM (en segundos)

Na Figura 4.2 pódense observar as aceleracións para cada número de elementos de procesado. Como se comentou previamente, a aceleración comeza a reducirse a partir dos oito nodos (128 elementos de procesado), polo que non consegue aproveitar as capacidades do hardware a partir dese punto. É dicir, ten mala escalabilidade. Porén, isto non supón un problema grave, xa que os tempos conseguidos son suficientemente baixos.

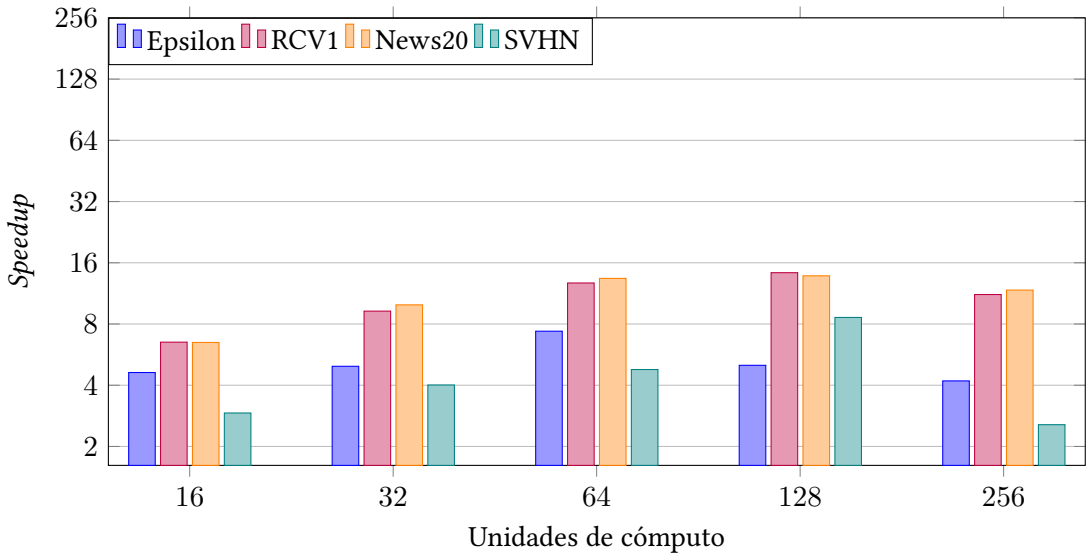


Figura 4.2: Aceleración obtida pola versión paralela de MIM incluída en Parallel-FST sobre a versión secuencial de FEAST

4.3.2 CondMI

Este método foi desenvolvido en [1] como unha optimización para tratar a selección de características como un problema de probabilidade condicional. Cabe recordar que é un algoritmo moi pesado, polo que foi necesario cambiar a configuración por nodo a un proceso sobre os 16 núcleos, ademais de non utilizar *Hyperthreading* con SVHN. Ademais, seleccionáronse só 50 características en lugar de 200 para cumprir coa restrición do clúster de execucións inferiores a 72h. Aínda con estas modificacións, segue a ser un dos algoritmos con maiores tempos de execución, como se mostra na Táboa 4.5.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	44429.6759	10064.1888	2919.3814	1463.5081	1632.4310	821.8625
RCV1	6221.1878	521.7783	273.2533	150.5470	89.3645	52.1172
News20	3798.9566	211.9196	110.0515	57.7233	30.5754	16.5788
SVHN	93245.8789	25778.2228	11999.9096	7513.7218	3432.9254	1679.4853
E2006	–	–	–	–	–	1782.8663

Táboa 4.5: Tempos do algoritmo CondMI (en segundos)

Na Figura 4.3 pódense ver os *speedups* para o algoritmo. Obsérvase unha particularidade que non se dá no resto de métodos: a aceleración, para un mesmo número de nodos, é moi diferente entre datasets, pero segue un crecemento máis ou menos constante para cada conxunto de datos particular. Observando os parámetros de cada conxunto, pódese intuír que isto se debe a se os datos conteñen máis características que mostras ou viceversa. Realmente este

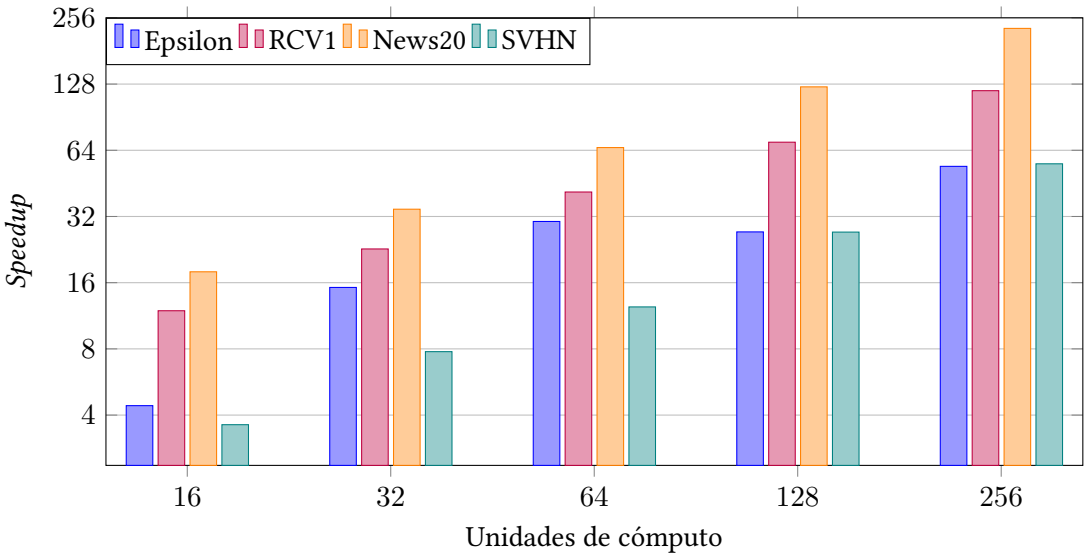


Figura 4.3: Aceleración obtida pola versión paralela de CondMI incluída en Parallel-FST sobre a versión secuencial de FEAST

fenómeno ocorre con todos os métodos, pero neste caso en particular vese máis acentuado.

4.3.3 BetaGamma

O método BetaGamma, tal como se comentou na Sección 2.4.3, engloba o espazo que recibe o seu mesmo nome, de forma que se pode utilizar para o cálculo de MIFS e CIFE. Na Táboa 4.6 amósanse os tempos de execución. As versións secuenciais tardan dende algo menos dunha hora ata máis de tres, e as execucións paralelas correspondentes redúcense a menos de 20 segundos e case un minuto, respectivamente.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	3571.1088	197.5616	105.7786	55.6688	34.0533	19.8030
RCV1	3924.5244	200.7787	101.7151	53.7469	29.6058	17.7222
News20	3936.0075	203.3170	103.0293	54.2713	29.9618	17.7814
SVHN	11156.8497	701.9341	352.8864	180.2394	97.7478	52.2868
E2006	–	–	–	–	–	1505.2784

Táboa 4.6: Tempos dos algoritmos no espazo BetaGamma (en segundos)

Como se pode observar na Figura 4.4, este é un método que escala moi ben de forma xeral para todos os conxuntos de datos. De feito, presenta superlinealidade para varios conxuntos ata os oito nodos, e para 16 alcanza eficiencias de ata o 87%.

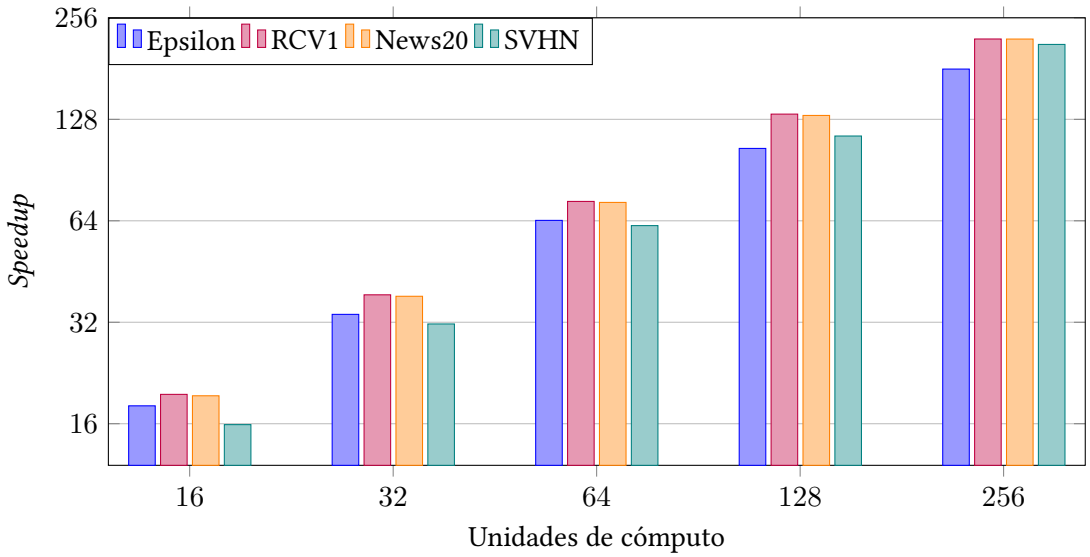


Figura 4.4: Aceleración obtida pola versión paralela de BetaGamma incluída en Parallel-FST sobre a versión secuencial de FEAST

4.3.4 JMI

Coa paralelización do método JMI tamén se conseguiron reducir significativamente os tempos de execución, como se pode observar na Táboa 4.7. A versión secuencial necesita dende media hora ata dúas horas e media, reducíndose coa versión paralela a menos de 15 segundos na maior parte dos datasets e a menos dun minuto para SVHN. O dataset máis grande (E2006) pode executarse no clúster nuns 13 minutos utilizando 16 nodos.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	1837.5858	108.2854	57.8770	30.8564	18.4884	12.0325
RCV1	1834.8695	94.8562	48.7740	27.1009	16.3776	10.6972
News20	1813.7040	96.3844	49.5091	27.3004	16.1237	10.6361
SVHN	8132.4649	424.9827	264.3953	142.9495	74.0361	40.9498
E2006	–	–	–	–	–	787.6408

Táboa 4.7: Tempos do algoritmo JMI (en segundos)

Como se pode observar na Figura 4.5, o método presenta boa escalabilidade, aínda que a partir de oito nodos a aceleración comeza a afastarse do número de elementos de procesado. Porén, para menos nodos (ata catro), presenta superlinealidade para varios datasets.

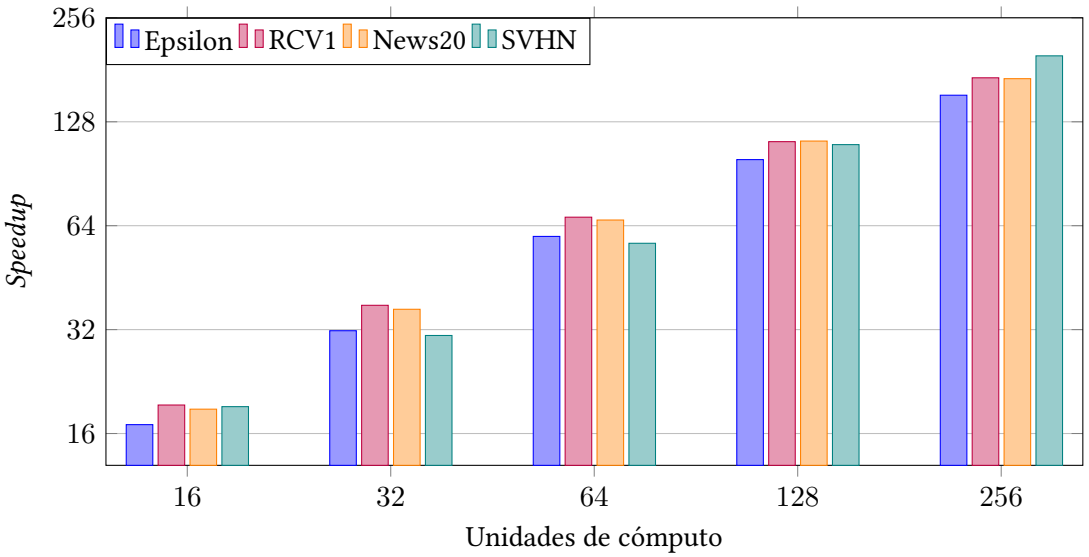


Figura 4.5: Aceleración obtida pola versión paralela de JMI incluída en Parallel-FST sobre a versión secuencial de FEAST

4.3.5 mRMR

A Táboa 4.8 amosa os tempos de execución para o método mRMR. Como se pode observar, as execucións orixinais comprendían tempos dende algo máis de 15 minutos ata 40. Coa execución da versión paralela en 16 nodos conséguese reducir estes tempos a menos de 20 segundos. No caso do conxunto E2006 pódese executar en menos de seis minutos empregando o clúster completo.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	1004.4851	55.5559	30.1156	17.2924	11.4081	8.3348
RCV1	1082.9073	57.7633	29.8370	17.9323	11.2156	7.1066
News20	1070.8058	58.0532	30.4536	17.9046	10.8822	7.1596
SVHN	2389.7123	130.1077	70.0608	40.6956	23.4888	17.6442
E2006	–	–	–	–	–	327.8149

Táboa 4.8: Tempos do algoritmo mRMR (en segundos)

Este método tamén presenta superlinealidade, pero só ata dous nodos, como se pode ver na Figura 4.6. A súa escalabilidade é boa, pero non tanto como a doutros métodos da libraría que teñen un tempo secuencial máis alto que mRMR. Como este método é relativamente rápido mesmo na súa versión secuencial, ao aumentar o número de nodos a eficiencia non crece de forma linealmente proporcional. De feito, a mellor eficiencia utilizando 16 nodos está en torno ao 60%.

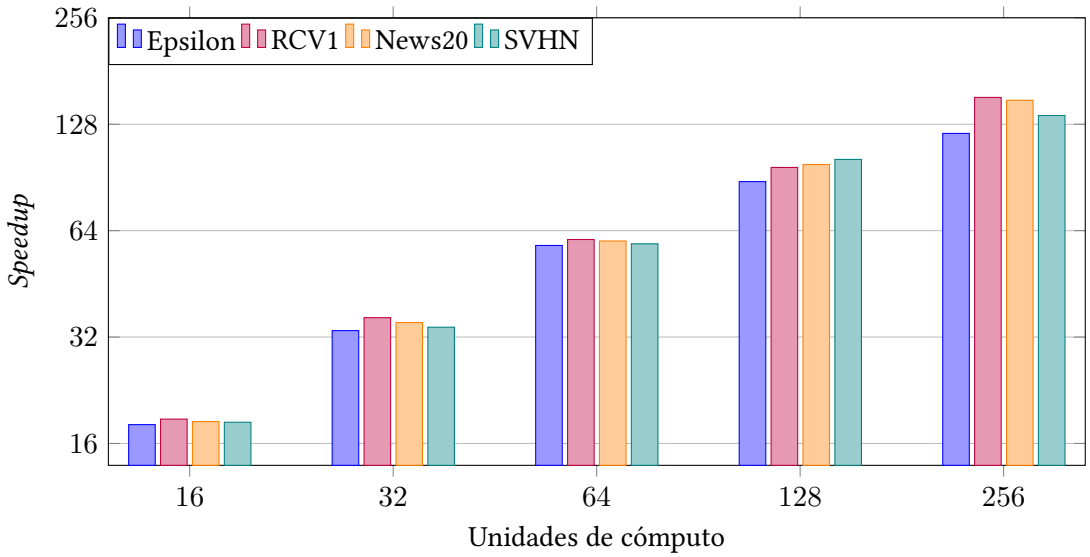


Figura 4.6: Aceleración obtida pola versión paralela de mRMR incluída en Parallel-FST sobre a versión secuencial de FEAST

4.3.6 CMIM

O algoritmo CMIM é o primeiro en avaliar que inclúe operadores de orde no cálculo das puntuacións, tal como se comentou na Sección 2.4.6. Cabe destacar que a implementación secuencial coa que se conta é a implementación rápida de [9]. Como se pode observar na Táboa 4.9, os tempos da versión paralela son moito peores que os da secuencial. Esta implementación contén un bucle *while*, que é o que lle dá a propiedade de "implementación rápida", xa que itera sobre o conxunto de características xa seleccionadas e pode deterse antes de analizalas todas, debido a que aplica unha suposición sobre a información mutua sobre as características restantes. Ao aplicar a paralelización, para que o bucle se deteña, a condición débese cumprir ao menos nunha característica das asignadas a cada elemento de procesado. Isto non se produce sempre en todos simultaneamente, polo que pode ser que uns elementos de procesado continúen a facer traballo non necesario, ralentizando a execución xeral. É por este motivo que se produce o empeoramento do tempo, non conseguindo aproveitar de forma óptima a arquitectura paralela do sistema. A vantaxe da implementación paralela incluída en Parallel-FST é que pode executar datasets grandes como E2006.

Na Figura 4.7 pódense ver as aceleracións das execucións paralelas, que, como consecuencia do empeoramento dos tempos, son todas inferiores a 0.25. Como se comentou anteriormente, a implementación secuencial é a rápida do artigo orixinal (que propón dúas implementacións: estándar e rápida). Por este motivo, os tempos secuenciais xa son bastante bos de por si, e por tanto recoméndase utilizar a implementación secuencial, salvo para datasets

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	149.6146	1510.7734	1314.9318	1156.2347	1008.6256	797.9973
RCV1	20.7887	124.4814	117.0114	108.7274	105.3173	101.2200
News20	17.4938	86.3293	72.6796	76.0579	81.5894	73.4239
SVHN	375.3060	8961.9295	9065.8332	8617.7012	6045.6973	3326.8385
E2006	–	–	–	–	–	127.3272

Táboa 4.9: Tempos do algoritmo CMIM (en segundos)

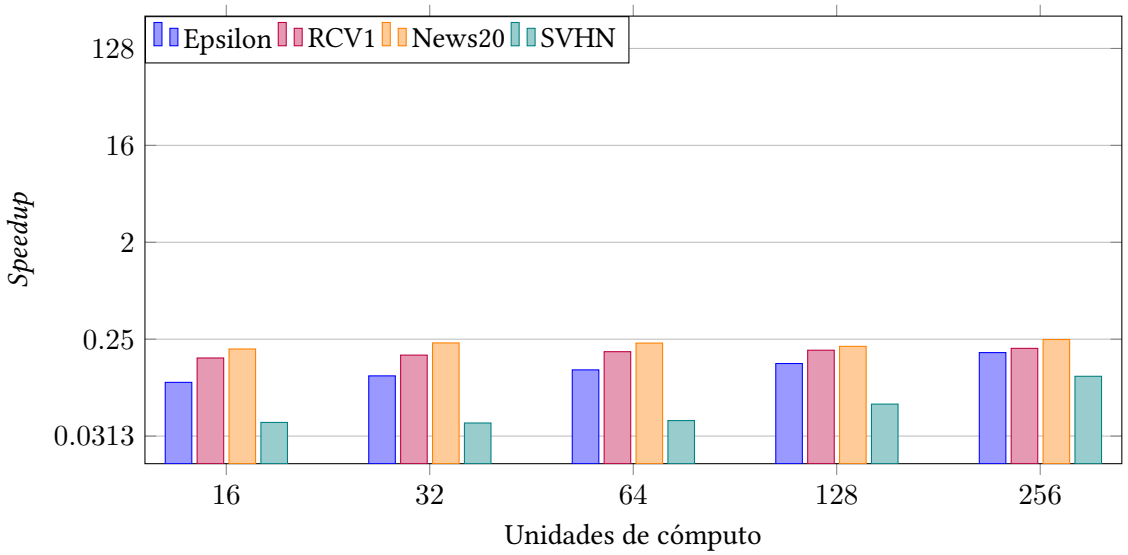


Figura 4.7: Aceleración obtida pola versión paralela de CMIM incluída en Parallel-FST sobre a versión secuencial de FEAST

grandes que ocasionen erros de memoria, como é o caso de E2006.

4.3.7 ICAP

Os tempos de execución deste método poden verse na Táboa 4.10. As versións secuenciais tardan dende pouco menos dunha hora ata algo máis de tres, e as execucións paralelas correspondentes redúcense a menos de 20 segundos e a case un minuto, respectivamente.

Como se pode observar na Figura 4.8, este é un método que escala moi ben para todos os datasets en xeral. Presenta superlinealidade para varios deles ata 8 nodos, e para 16 alcanza eficiencias de ata o 87%.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	3521.3014	193.1773	101.9894	54.7378	33.8858	19.1572
RCV1	3923.7314	200.8248	101.8808	53.8442	29.6778	17.7062
News20	3924.6220	203.2614	102.9128	54.1602	29.8708	17.5622
SVHN	11023.4940	577.0990	359.2827	181.3007	98.7142	55.7405
E2006	–	–	–	–	–	1462.6579

Táboa 4.10: Tempos do algoritmo ICAP (en segundos)

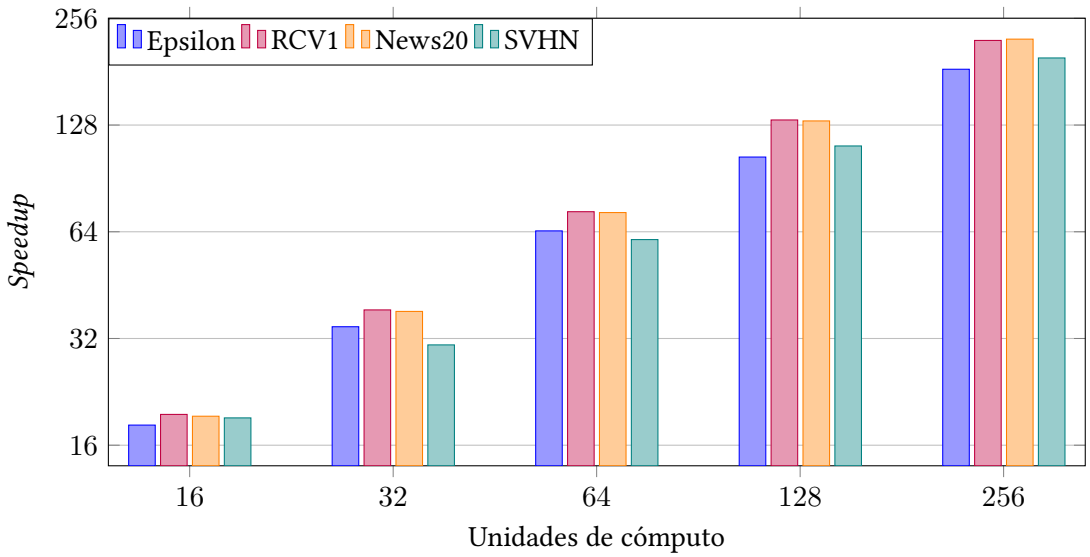


Figura 4.8: Aceleración obtida pola versión paralela de ICAP incluída en Parallel-FST sobre a versión secuencial de FEAST

4.3.8 DISR

A Táboa 4.11 mostra os tempos das execucións do método DISR. Como se pode observar na mesma, conseguíronse reducir os tempos, que na versión secuencial abarcaban dende os 45 minutos ata as tres horas e media, a menos de 20 segundos e algo máis dun minuto, respectivamente. A implementación paralela permitiu ademais executar o dataset E2006 (empregando 19 minutos), o que non foi posible coa versión secuencial.

Na Figura 4.9 pódense observar as aceleracións. O método presenta boa escalabilidade para todos os datasets en xeral, aínda que a partir dos oito nodos a aceleración comeza a afastarse do óptimo. Por outra parte, presenta superlinealidade para algúns datasets ata catro nodos.

Dataset	Secuencial	1N	2N	4N	8N	16N
Epsilon	2900.9330	162.8393	85.0760	45.5118	25.9510	16.1518
RCV1	2849.8846	144.9506	74.2375	39.8059	22.5821	14.2446
News20	2808.4364	148.8027	75.5602	40.3279	23.0448	14.4433
SVHN	12785.0935	850.9628	432.3944	215.8255	119.6403	68.0067
E2006	–	–	–	–	–	1123.0589

Táboa 4.11: Tempos do algoritmo DISR (en segundos)

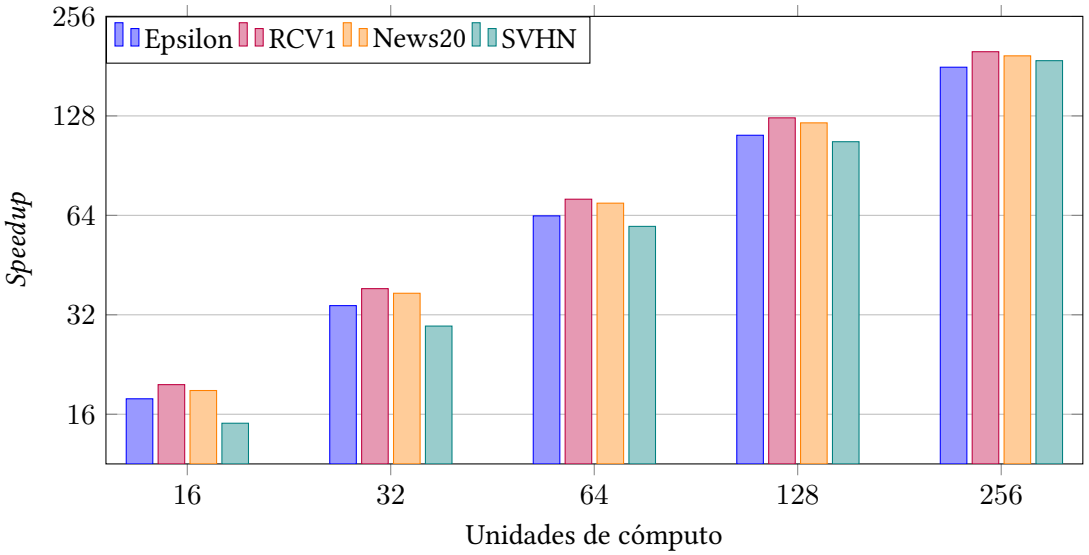


Figura 4.9: Aceleración obtida pola versión paralela de DISR incluída en Parallel-FST sobre a versión secuencial de FEAST

4.4 Conclusións xerais

Dende un punto de vista xeral, pódese concluír que os resultados acadados son moi satisfactorios, por dous motivos:

- É posible executar os métodos de selección de características con datasets de gran tamaño, que non caberían na memoria dun só nodo, como é o caso de E2006.

- Para os demais datasets é posible executar todos os métodos en tempos moi razoables. A excepción de CondMI, que conta cunha maior complexidade, e de CMIM, debido á súa excepcionalmente rápida implementación orixinal, é posible seleccionar 200 características destes datasets practicamente en menos dun minuto utilizando a versión paralela en 16 nodos (sendo o método DISR co dataset SVHN o único que supera o minuto, en concreto, 68 segundos).

Grazas ao emprego de datasets moi variados podemos comparar como se comportan os métodos en escenarios diferentes. Algo que teñen en común todos os métodos, incluídos aqueles que non aproveitan ben as técnicas de paralelización (MIM e CMIM), é que as aceleracións son, na maioría dos casos, superiores para os datasets RCV1 e News20. De feito, no caso de CondMI, chegan incluso a duplicarse. Isto débese a que son os dous datasets con moitas máis características que mostras (ver Táboa 4.2). Como a implementación paralela distribúe as características entre os procesos, cando temos máis características o reparto é mellor entre todos. Ademais, como os procesos acceden secuencialmente a todas as mostras dunha mesma característica, os accesos a memoria son máis eficientes cando o número de características é menor. A boa noticia é que na actualidade a maior parte de datasets *big data* dispoñibles teñen moitas máis características que mostras (por exemplo, datos biolóxicos onde o máis difícil é atopar individuos que se ofrezan a pasar por un estudo).

Capítulo 5

Planificación e custos

Neste capítulo achégase a planificación detallada da organización do Traballo Fin de Grao. En primeiro lugar, coméntanse as fases nas que se dividiu, e posteriormente especifícase a duración e custo de cada unha.

5.1 Planificación do proxecto

Durante a realización deste Traballo Fin de Grao seguiuse un modelo de desenvolvemento incremental, no que se foron incluíndo pouco a pouco novos métodos paralelos á libraría, intercalando, polo tanto, fases de deseño e implementación con fases de avaliación para cada un dos métodos.

5.1.1 Fase 1: Estudo e aproximación ao campo da selección de características e aos algoritmos da libraría orixinal

En primeiro lugar levouse a cabo unha fase de documentación para adentrarse no campo de estudo. Para isto, foi necesaria a búsqueda e lectura de artigos. Tendo en conta que este Traballo Fin de Grao se centra na libraría FEAST, o primeiro artigo estudado foi [1], do que, a continuación, se analizou o código fonte co fin de comprender o seu funcionamento. Posteriormente, tras a lectura de máis investigacións [15, 23], decidiuse que as técnicas e optimizacións paralelas se realizarían utilizando MPI e tecnoloxías multifío.

Por outra parte, como toma de contacto con FEAST, desenvolveuse unha primeira optimización *adhoc* para o método CMIM, escollido arbitrariamente. Con isto foi posible comprender o funcionamento interno dos algoritmos, así como as medidas de información que utilizan. A maiores, as probas para esta primeira implementación permitiron detectar os problemas de memoria para datasets relativamente pequenos, o que motivou o desenvolvemento da compresión de rango descrita na Sección 3.5.2.

5.1.2 Fase 2: Deseño xeral da libraría

Unha vez comprendido o funcionamento do algoritmo CMIM, o seguinte paso foi o desenvolvemento dunha implementación máis robusta, seguida das correspondentes probas de rendemento. Porén, os tempos de execución non melloraron no caso de CMIM, polo que foi necesario investigar a causa. Este problema debíase a que a estrutura do algoritmo non lle permite aproveitar as vantaxes da descomposición de dominio adoptada (o que se explicou previamente na Sección 4.3.6).

Para continuar co desenvolvemento, escolleuse un novo método sobre o que traballar, neste caso DISR. Coa previsión de engadir todos os algoritmos da libraría orixinal ao proxecto, decidiuse deseñar de novo o programa para adaptalo a estas futuras ampliacións. Ademais, neste momento os arquivos que consituían o proxecto comezaban a aumentar en número e tamaño, polo que se decidiu integrar CMake para simplificar a xestión da compilación.

5.1.3 Fase 3: Tratamento de cada método en particular

Como se comentou previamente, a idea era engadir pouco a pouco as versións paralelas de todos os métodos de [1]. Polo tanto, a os pasos que se seguiron para cada un deles foron os seguintes:

1. Análise do funcionamento e das métricas utilizadas.
2. Deseño da versión multifío.
3. Deseño da versión híbrida engadindo o uso de MPI.
4. Implementación das técnicas de paralelización e optimización, para as versións normais e *weighted*.
5. Probas de validación para asegurar que os resultados coinciden cos da versión secuencial.
6. Probas menores de avaliación do rendemento, coa idea de detectar erros na implementación ou incompatibilidades, como no caso do método CMIM.

5.1.4 Fase 4: Avaliación do rendemento

Unha vez que se contaba coas versións paralelas, o seguinte paso era avaliar o rendemento e a escalabilidade. Como se viu en capítulos anteriores, estas probas consistiron en dúas partes: en primeiro lugar, era necesario atopar a mellor combinación de procesos de MPI e fíos para cada nodo, e despois medir os tempos das execucións con esa configuración en máis nodos. Durante as probas do primeiro algoritmo atopouse que un dataset en formato

disperso causaba que se esgotase a memoria dun nodo ao expandilo, polo que foi necesario implementar a lectura semi-distribuída (descrita na Sección 3.5.1).

Nun principio, esta tarefa realizaríase en paralelo á inclusión dos métodos á libraría, polo que, despois de rematar co primeiro, desenvolvéronse os *scripts* para lanzar as probas no clúster. Porén, ao levar implementada aproximadamente a metade dos algoritmos xurdiu un imprevisto, xa que se produciu unha actualización do software do clúster que supuxo o cambio do sistema operativo e do sistema de planificación. Debido a este problema, decidiuse rematar primeiro o desenvolvemento de todas as versións paralelas e, posteriormente, realizar todas as probas xuntas unha vez rematase a actualización do clúster. Ademais, foi necesario realizar cambios en todos os *scripts* para adaptalos ao novo sistema de planificación.

5.1.5 Fase 5: Documentación e escritura da memoria

Ao longo do ciclo de vida do proxecto fóronse documentando todos os pasos seguidos. Durante as primeiras etapas, recolléronse anotacións cos obxectivos, problemas e resultados de cada día de traballo. Aínda así, aspectos máis destacables como a compresión de rango contaron cunha documentación máis detallada. Durante as etapas máis avanzadas, como as de deseño e implementación, a documentación foi máis extensa (diagramas de clases, diagramas de comunicacións entre procesos, etc.). O último paso foi a compilación de toda a documentación e dos resultados nesta memoria.

5.2 Métricas do proxecto

Ademais da organización do proxecto, tamén se ten en conta a duración e os recursos, así como os seus custos teóricos.

5.2.1 Duración

A duración do proxecto foi de oito meses, dende setembro de 2019 ata abril de 2020. Cabe destacar que o proxecto se suspendeu en diversas ocasións pola falta de tempo motivada polos estudos en curso, e que ademais se produciron certos cambios debido a imprevistos. Na planificación inicial tivéronse en conta as suspensións por estudos e estimouse que o proxecto contaría cun esforzo aproximado de 300hp (horas $\times$ persoa), a realizar en sete meses (oito, pero sen dedicación durante un mes de exames). Porén, os imprevistos afectaron da seguinte maneira:

- Actualización do clúster: como se comentou previamente, esta actualización supuxo un cambio no sistema operativo, que non influíu no transcurso do proxecto, e tamén un

cambio no sistema de planificación, que invalidou todos os *scripts* de probas desenvolvidos ata o momento, polo que foi necesario refacelos. Isto supuxo que o esforzo desta tarefa aumentase de 20hp a 25hp, e que ademais se pospuxese ata o fin da implementación das versións paralelas.

- Execucións con consumo de memoria alto para datasets pequenos: ao detectarse este problema, foi necesario introducir unha nova tarefa para tentar solucionalo. En concreto, esta tarefa concluíu co desenvolvemento da "compresión de rango", e supuxo un esforzo a maiores de 30hp.

- Esgotamento de memoria por datasets de gran tamaño: este problema solucionouse desenvolvendo a técnica da "lectura semi-distribuída", para o que foi necesario engadir unha nova tarefa cun esforzo a maiores de 15hp.

Como se pode observar, estes imprevistos aumentaron o esforzo do proxecto aproximadamente en 50hp. Isto poderíase solucionar estendendo a duración do mesmo arredor de 50hp coa mesma dedicación que a planificada (o que equivalería a unhas catro semanas a maiores). Porén, coa intención de axustarse o máximo posible ás datas da primeira planificación, decidiuse aumentar a dedicación durante aquelas fases nas que foi necesario.

Na Táboa 5.1 amósase unha descomposición destas tarefas, e na Figura 5.1 un diagrama de Gantt para ilustrar as tarefas no tempo e as dependencias entre elas.

As dependencias máis relevantes que se atoparon entre tarefas foron as seguintes, agrupadas por tipo:

- **Dependencias Fin-Comezo (F-S)**. Son aquelas que indican que a tarefa sucesora non pode comezar ata que a predecesora remate. En concreto, as relacións que se identificaron deste tipo son:

  - **Tarefa a** $\rightarrow$ **Tarefas c** e **d**. A **Tarefa a** marca o inicio do proxecto co estudo da libraría orixinal, polo tanto, ata que se comprende o seu funcionamento non ten sentido investigar en áreas similares (**Tarefa c**) nin comezar co deseño do sistema (**Tarefa d**).

  - **Tarefa f** $\rightarrow$ **Tarefas g**, **h**, **i**, **l**, **m**, **n** e **o**. Na **Tarefa f** realízase a paralelización do método DISR que, ao ser a que se desenvolve xunto co deseño da estrutura da libraría (**Tarefa d**), debe completarse antes de comezar as demais paralelizacións (**Tarefas g**, **h**, **i**, **l**, **m**, **n** e **o**). Ademais, por este motivo estímase que a duración será maior que para o resto de paralelizacións.

  - **Tarefas p** e **q** $\rightarrow$ **Tarefa r**. A **Tarefa r** constitúe a execución das probas de rendemento e a recolección de resultados. Como se explica no Capítulo 4, antes de

Fase	Tarefa	Tempo (h)
1	a. Estudo da libraría orixinal	20
	b. Identificación e desenvolvemento da "compresión de rango"	30
2	c. Investigación de artigos en áreas similares	15
	d. Deseño da estrutura da libraría e integración con CMake	30
	e. Identificación e desenvolvemento da "lectura semi-distribuída"	15
3	f. Paralelización DISR	15
	g. Paralelización JMI	5
	h. Paralelización CondMI	5
	i. Paralelización CMIM	5
	l. Paralelización mRMR	5
	m. Paralelización MIM	5
	n. Paralelización BetaGamma	5
	o. Paralelización ICAP	5
4	p. Desenvolvemento de *scripts* para as probas de rendemento	25
	q. Busca da mellor configuración para un nodo	20
	r. Probas de rendemento e obtención de resultados	40
5	s. Elaboración da documentación de deseño e implementación	10
	t. Elaboración da documentación da interpretación dos resultados	25
	u. Escritura da memoria	70

Táboa 5.1: Horas invertidas en cada tarefa

lanzar as probas en varios nodos é necesario coñecer a mellor configuración para un só (**Tarefa q**), ademais de ter implementados os *scripts* para a execución das probas (**Tarefa p**).

- **Dependencias Comezo-Comezo (S-S)**. Son aquelas que indican que a tarefa sucesora non pode comezar ata que a predecesora comece. Habitualmente, aínda que non sexa o seu significado real, utilízase para modelar tarefas que comezan á vez. En concreto, as relacións que se identificaron deste tipo son:

  - **Tarefa b** $\rightarrow$ **Tarefa s**. Ao longo da **Tarefa s** elabórase a documentación do deseño e implementación, polo que non pode comezar ata que non se empece a deseñar ou a implementar (neste caso a **Tarefa b**, xa que constitúe o desenvolvemento da compresión de rango).
  - **Tarefas c** e **d** $\rightarrow$ **Tarefa f**. Na **Tarefa f**, como se comentou previamente, realízase a primeira paralelización dun método integrada na estrutura da libraría paralela. Para isto, é necesario que xa se iniciasen tanto o deseño da estrutura da libraría paralela (**Tarefa d**) como o estudo de artigos de investigacións similares (**Tarefa c**). De feito, o ideal sería que xa se avanzase algo nesas tarefas antes de comezar coa **Tarefa f**.

  - **Tarefas d, p, q** e **t**. Estas tarefas están relacionadas coas probas de rendemento, e dependen entre si desta forma: a interpretación dos resultados das probas (**Tarefa t**) non pode comezar ata que se inicien ditas probas (en concreto, as da búsqueda da mellor configuración para un nodo, na **Tarefa q**); á súa vez, as probas non se poden realizar ata que se conte con *scripts* para lanzalas (ver primeira parte da **Tarefa p** no diagrama de Gantt da Figura 5.1); por último, non se pode iniciar o desenvolvemento dos *scripts* ata que comece o desenvolvemento do sistema na **Tarefa d** (idealmente, ata que se avance un pouco).

- **Dependencias Fin-Fin (F-F)** Son aquelas que indican que a tarefa sucesora non pode rematar ata que remate a predecesora. En concreto, as relacións que se identificaron deste tipo son:

  - **Tarefas b**, **d**, **e** e **f**. A **Tarefa d** constitúe o deseño da libraría paralela, polo que non pode rematar ata que finalicen o desenvolvemento das solucións da compresión de rango (**Tarefa b**) e da lectura semi-distribuída (**Tarefa e**). Por outra parte, a paralelización do primeiro método (**Tarefa f**) precisa que a libraría conte cunha estrutura, polo que non pode rematar ata que conclúa a **Tarefa d**.
  - **Tarefas g**, **h**, **i**, **l**, **m**, **n** e **o** $\rightarrow$ **Tarefa q** e **s**. As **Tarefas g**, **h**, **i**, **l**, **m**, **n** e **o** constitúen as paralelizacións dos métodos (a excepción do primeiro). Estas implementacións precísanse para a busca da mellor configuración para un nodo (**Tarefa q**), tarefa que non pode finalizar ata que se teñen todas as versións paralelas. Por outra parte, é necesario documentar as decisións de deseño e implementación das mesmas, polo que a **Tarefa s** tampouco pode finalizar ata que rematen.
  - **Tarefa r** $\rightarrow$ **Tarefa t**. A elaboración da documentación da interpretación dos resultados (**Tarefa t**) contén tanto os resultados da mellor configuración para un nodo como os das probas de rendemento (**Tarefa r**), polo que non se pode concluír a documentación dos resultados mentres non se teñan todos.
  - **Tarefas s** e **t** $\rightarrow$ **Tarefa u**. Nesta memoria recóllense, entre outras, a documentación do deseño e implementación da libraría (**Tarefa s**) e a interpretación dos resultados das probas de rendemento (**Tarefa t**), polo que é necesario que finalicen antes de poder rematar a elaboración da memoria (**Tarefa u**).

Por último, cabe destacar que as **Tarefas b** e **e** xurdiron para solucionar problemas que se detectaron durante a realización do proxecto, polo que non se podían prever. Por este motivo, non se modelou ningunha relación que achegue información sobre cando comezan. Porén, se se coñecesen dende un principio, o apropiado sería unha relación Comezo-Comezo dende a tarefa na que se detectan (**Tarefas a** e **d**, respectivamente), para indicar que non se pode comezar a buscar unha solución antes de que comece a tarefa na que xorde o problema.

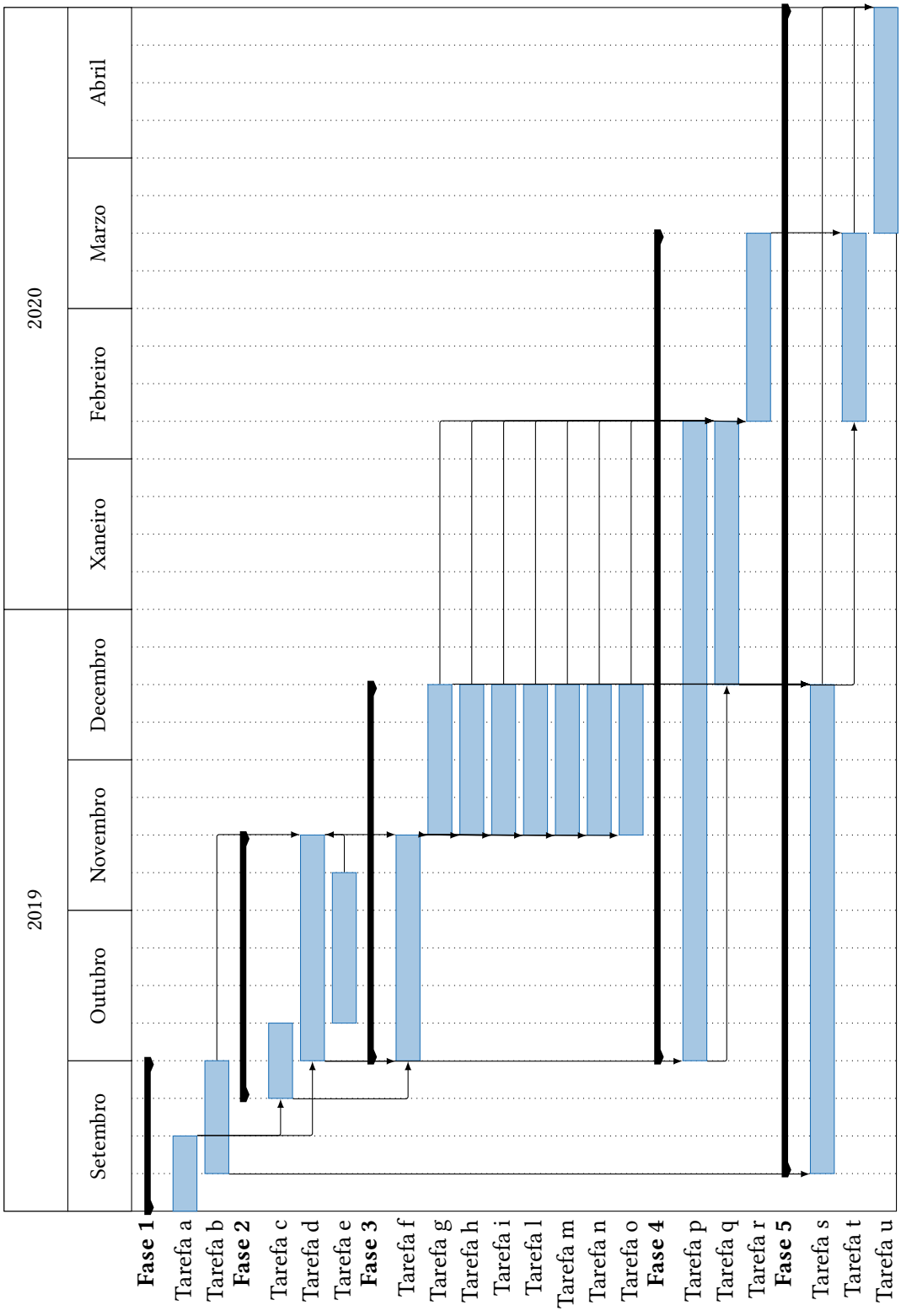


Figura 5.1: Diagrama de Gantt do desenvolvemento do proxecto

5.2.2 Presuposto

No desenvolvemento deste proxecto participaron tres persoas: o estudante, encargado do desenvolvemento do código e da obtención dos resultados, e dous titores, encargados da proposta dos temas do proxecto e da supervisión do mesmo.

Na Táboa 5.2 amósase unha relación aproximada das horas que empregou cada recurso. Como o desenvolvemento do proxecto sufriu suspensións e a dedicación foi variable, os datos son aproximados. Estimáronse para o estudante a unha media de 12.5 horas semanais (entre dúas e tres horas diarias, e tendo en conta que o proxecto estivo detido aproximadamente un mes polos exames do primeiro cuadrimestre). O número de horas dos titores estimáronse en función do número de reunións presenciais e non presenciais, conversas a través do correo electrónico e revisións de documentos.

Recurso	**Dedicación (h)**
Estudante	350
Titor 1	30
Titor 2	20

Táboa 5.2: Horas invertidas por cada recurso no proxecto

Por último, na Táboa 5.3 amósase a relación do custo aproximado da hora de cada recurso, e o custo total aproximado do proxecto.

Recurso	**Custo por hora (€/h)**	**Horas**	**Custo (€)**
Estudante	40	350	14000
Titores	60	50	3000
Total			17000

Táboa 5.3: Custos totais do proxecto

Capítulo 6

Conclusións

Neste capítulo achéganse as conclusións alcanzadas coa realización deste Traballo Fin de Grao, así como a súa relación coas materias da titulación e as liñas de traballo futuro polas que se podería continuar.

6.1 Conclusións

O obxectivo principal deste Traballo Fin de Grao consistiu no desenvolvemento dunha libraría paralela de algoritmos de selección de características, o que permitiu, por unha parte, proporcionar un produto de código libre utilizable polo público e, por outra, afondar tanto no campo da selección de características como no da programación paralela.

Observando os resultados obtidos no Capítulo 4, pódese concluír que os obxectivos cumpríronse de forma máis que satisfactoria, xa que as solucións propostas, na súa maioría, achegan moi bos resultados en termos de aceleración e escalabilidade. Como se comentou en capítulos anteriores, danse dúas excepcións (os métodos MIM e CMIM) que non se benefician das técnicas de paralelización, o cal non constitúe un problema debido a que as súas versións secuenciais son suficientemente rápidas. De feito, con sete dos oito métodos é posible seleccionar 200 características de datasets de tamaño considerable en 16 nodos (256 elementos de procesado) do clúster "Plutón" en menos de 70 segundos. O oitavo método, CondMI, non se agrupa cos anteriores, xa que a súa maior complexidade causa tempos de execución moi altos. Porén, tamén se beneficia das técnicas de paralelización, chegando a execucións ata 229 veces máis rápidas que na versión secuencial correspondente.

Outro aspecto destacable é a posibilidade de aplicar a selección de características sobre datasets con tamaños superiores ás capacidades de memoria dun só nodo. É dicir, non só a execución é máis rápida en Parallel-FST que na orixinal FEAST, senón que a versión paralela é capaz de procesar datasets que non era posible executar coa versión secuencial xa que xurdían problemas de memoria.

Tamén relacionado cos datasets, as técnicas de computación paralela presentan mellor escalabilidade ao traballar con aqueles con máis características que mostras, o que se pode ver como unha vantaxe debido a que a maior parte dos datasets actuais teñen esa estrutura. Como exemplos temos os datos biolóxicos, nos que é máis fácil tomar moitas medicións que atopar voluntarios para estudo, ou os datos de linguaxe, onde o uso de bigramas e trigramas ocasiona unha explosión do número de características (que se potencia máis incluso co uso de n-gramas de máis elementos).

A realización deste Traballo Fin de Grao permitiume expandir os meus coñecementos nas áreas de *big data*, intelixencia artificial e HPC. Inicialmente, os meus coñecementos nestes campos eran baixos, polo que foi necesario invertir bastante tempo no estudo, o que me achegou experiencia que considero que me será de utilidade no futuro. Ademais, co desenvolvemento de Parallel-FST mellorei as miñas capacidades no uso de certas ferramentas e tecnoloxías, como C++ ou MPI, e tamén aprendín algunhas novas, como CMake. En xeral, considero que a realización deste proxecto me permitiu mellorar as miñas habilidades de desenvolvemento software, así como comezar a introducirme no ámbito da investigación.

6.2 Relación coa titulación

O desenvolvemento deste proxecto permitiume poñer en práctica algúns dos coñecementos obtidos ao longo da carreira. En concreto, para o deseño da arquitectura da libraría paralela utilizouse unha linguaxe que admitía orientación a obxectos, polo que foron útiles varios patróns de deseño aprendidos na materia "Deseño Software". Ademais, ao facer uso de MPI e tecnoloxías multifío, utilizáronse técnicas que se explican en "Concorrencia e Paralelismo". Por outra parte, a implementación e mantemento do código beneficiáronse de coñecementos adquiridos a través de moitas materias, isto é, as bases da programación estruturada e o uso de sistemas de control de versións. Cabe destacar que, aínda que as bases da programación se estudan en "Programación I e II", estes coñecementos vanse refinando ao ser necesario aplicalos nos traballos doutras materias. Por último, neste proxecto tentouse planificar seguindo un ciclo de desenvolvemento clásico, estudado en "Xestión de Proxectos", aínda que ao ser un proxecto de investigación, a estimación da duración das tarefas non pode realizarse de maneira moi precisa.

A maiores, os coñecementos específicos adquiridos ao cursar a Mención en Computación tamén influíron positivamente neste proxecto. Por exemplo, na materia "Recuperación da Información" explícanse e estúdanse varias métricas de información, o que simplificou o entendemento destes conceptos ao inicio do traballo. Por outra parte, en "Aprendizaxe Automática" achégase unha pequena introdución tanto á selección de características como aos modelos que se poden beneficiar da mesma. Tamén se estudan as árbores de decisión que, como se

comentou no seu momento, aplican a selección de características no proceso de construción do modelo (i.e. métodos *wrapper*).

6.3 Traballo futuro

En primeiro lugar, Parallel-FST desenvolveuse co obxectivo de ser facilmente ampliable e mantible. Por este motivo, un dos obxectivos futuros é engadir novos algoritmos, e tamén permitir a lectura de datos en novos formatos, se isto é necesario. Por outra parte, a modificación dos algoritmos non ten por que ser en "amplitude" (engadir máis), senón que se pode centrar en introducir melloras para os xa presentes. Por exemplo, poderíanse considerar novas opcións para a distribución do cómputo, como pode ser o uso de GPUs ou outro hardware específico, tanto utilizando aproximacións homoxéneas como híbridas.

Por último, ao liberarse a libraría como código aberto, preténdese posibilitar que calquera desenvolvedor interesado contribúa con calquera outra mellora ou modificación froito de requisitos específicos que non se tiveron en conta durante o deseño.

Apéndices

Apéndice A

Guía de uso de Parallel-FST

O obxectivo desta guía é o de achegar unha explicación da instalación, configuración e execución de Parallel-FST.

A.1 Requisitos

O proxecto fai uso de varias librarías e programas, tanto para a configuración e compilación como para a execución. En concreto, as versións utilizadas foron as seguintes:

- **GCC** v5.4.0
- **make** v4.1
- **CMake** v2.6
- **MPI** (só é necesaria unha):
  - **MPICH** v3.2
  - **OpenMPI** v3.1.0
- **Git** (opcional)

Cabe destacar que é posible a compilación e execución con versións anteriores, pero non se garanten uns resultados correctos.

As librarías FEAST e MIToolbox están incluídas no proxecto, de forma que non é necesario telas previamente instaladas.

A.2 Compilación

O proxecto deseñouse para que non sexa necesaria unha instalación a nivel de sistema, senón que se poida compilar e executar. Os pasos a seguir para obter un programa executable son os seguintes:

1. Descarga. En primeiro lugar, é necesario obter os arquivos do proxecto. Isto pódese realizar cunha descarga directa dende a web do proxecto, ou opcionalmente clonándoo con Git.

2. Configuración. Este paso fai uso de CMake para preparar os arquivos para a compilación, e detectar erros, como a falta de librarías. Para isto, situámonos no directorio raíz do proxecto e lanzamos o *script* "*configure*" co comando `./configure`.

3. Compilación. Este paso utiliza "*make*" para crear as librarías dinámicas a partir de FEAST e MIToolbox, e posteriormente enlazalas co executable. Dende o directorio do proxecto lanzamos o comando `make`.

4. Limpeza (opcional). Opcionalmente, achégase un *script* "*clean.sh*" para eliminar os arquivos xerados durante a configuración e compilación. Cabe destacar que só se eliminan os arquivos innecesarios, de forma que as librarías e o executable seguen a funcionar. Lánzase dende o directorio do proxecto con `./clean.sh`.

A.3 Execución

Parallel-FST débese lanzar cos comandos de execución de MPI (habitualmente, `mpiexec` -estándar- ou `mpirun`). Polo tanto, o comando que define a execución ten a seguinte forma:

```
mpiexec -n <numProcs> ./Parallel-FST <options>
```

onde `numProcs` é o número de procesos de MPI a lanzar, e `options` é unha lista dos seguintes argumentos (obrigatorios, salvo que se indique o contrario):

`-a` `<algorithm>`. Criterio de selección de características a utilizar. `algorithm` é unha cadea de texto das seguintes: "mRMR_D", "CMIM", "JMI", "DISR", "ICAP", "CondMI", "MIM" ou "BetaGamma".

`--original`. Flag que indica se se debe utilizar a versión orixinal (secuencial) do algoritmo. Só se pode utilizar cando se lanza un único proceso cun único fío.

`-i` `<inputFile>`. Dataset sobre o que aplicar a selección de características. `inputFile` indica o *path* do arquivo que contén o dataset. Actualmente, sopórtanse os formatos ARFF, CSV e LibSVM.

`-w` `<weightsFile>` (opcional). Permite o uso das versións *weighted* dos algoritmos cando se especifica o arquivo que contén os pesos. `weightsFile` é o *path* dese arquivo, e o seu formato debe ser o seguinte: unha liña por mostra co peso da mostra

correspondente, na mesma orde que no dataset. Cabe destacar que é obrigatorio que coincidan o número de pesos neste arquivo e o de mostras no dataset.

-o <outputFile> . Arquivo no que escribir os resultados. outputFile é o *path* de dito arquivo. O formato de saída é o seguinte: separados por un espazo en branco, o índice no *dataset* de entrada e a puntuación obtida segundo o criterio utilizado para cada característica, cunha liña por característica seleccionada.

-s <numSelecFeat> . Permite especificar o número de características a seleccionar con numSelecFeat .

-d <numBins> (opcional). Indica se se debe aplicar a discretización con *binning*, onde o número de *bins* a utilizar especifícase con numBins .

-t <numTh> (opcional). Permite especificar o número de fíos numTh que debe lanzar cada proceso de MPI con numTh . Se non se especifica, utilízase o número de núcleos detectado polo sistema operativo.

-m (opcional). Flag que activa a aplicación da técnica da compresión de rango.

--beta <b> **--gamma** <g> (só para BetaGamma). Cando se utiliza o método *BetaGamma* permite especificar os valores de beta (b) e gamma (g).

-h (opcional). Imprímese unha mensaxe de axuda sobre o uso de Parallel-FST e o formato e explicación dos argumentos de liña de comandos.

Relación de Acrónimos

ARFF *Attribute Relation File Format.*

CIFE *Conditional Infomax Feature Extraction.*

CMIM *Conditional Mutual Information Maximisation.*

CondMI *Conditional Mutual Information.*

CPU *Central Processing Unit.*

CR *Compresión de Rango.*

CSV *Comma Separated Values.*

CUDA *Compute Unified Device Architecture.*

DISR *Double Input Symmetrical Relevance.*

FEAST *FEAture Selection Toolbox.*

GPU *Graphics Processing Unit.*

HPC *High Performance Computing.*

ICAP *Interation CAPping.*

JMI *Joint Mutual Information.*

LAN *Local Area Network.*

LDA *Linear Discriminant Analysis.*

LibSVM *Library for Support Vector Machines.*

MIFS *Mutual Information Feature Selection.*

MIM *Mutual Information Maximisation.*

MIToolbox *Mutual Information Toolbox.*

MPI *Message Passing Interface.*

mRMR *Max-Relevance Min-Redundancy.*

NAS *Network Attached Storage.*

OpenCL *Open Computing Language.*

Parallel-FST *Parallel Feature Selection Toolbox.*

PCA *Principal Component Analysis.*

RAM *Random Access Memory.*

SMT *Simultaneous MultiThreading.*

SPMD *Single Program Multiple Data.*

Bibliografía

[1] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional Likelihood Maximisation: A Unifying Framework for Information Theoretic Feature Selection," *J. Mach. Learn. Res.*, vol. 13, pp. 27–66, 2012.

[2] D. Dua and C. Graff, "UCI Machine Learning Repository," 2017. [En liña]. Dispoñible en: http://archive.ics.uci.edu/ml

[3] P. Cunningham, "Dimension Reduction," *Lecture Notes in Applied and Computational Mechanics*, pp. 91–112, 2008.

[4] D. D. Lewis, "Feature Selection and Feature Extraction for Text Categorization," in *Proceedings of the Workshop on Speech and Natural Language*, ser. HLT '91. USA: Association for Computational Linguistics, 1992, pp. 212–217. [En liña]. Dispoñible en: https://doi.org/10.3115/1075527.1075574

[5] R. Battiti, "Using Mutual Information for Selecting Features in Supervised Neural Net Learning," *IEEE Trans. Neur. Netw.*, vol. 5, no. 4, pp. 537–550, 1994. [En liña]. Dispoñible en: https://doi.org/10.1109/72.298224

[6] D. Lin and X. Tang, "Conditional Infomax Learning: An Integrated Framework for Feature Extraction and Fusion," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ser. ECCV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 68–82. [En liña]. Dispoñible en: https://doi.org/10.1007/11744023_6

[7] H. H. Yang and J. Moody, "Data Visualization and Feature Selection: New Algorithms for Nongaussian Data," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 687–693.

[8] H. Peng, F. Long, and C. Ding, "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy," *IEEE Trans. Pattern*

Anal. Mach. Intell., vol. 27, no. 8, pp. 1226–1238, 2005. [En liña]. Dispoñible en: https://doi.org/10.1109/TPAMI.2005.159

[9] F. Fleuret, "Fast Binary Feature Selection with Conditional Mutual Information," *J. Mach. Learn. Res.*, vol. 5, pp. 1531–1555, 2004.

[10] A. Jakulin, "Machine Learning Based on Attribute Interactions," Ph.D. dissertation, University of Ljubljana, Slovenia, 2005.

[11] P. E. Meyer and G. Bontempi, "On the Use of Variable Complementarity for Feature Selection in Cancer Classification," in *Proceedings of the 2006 International Conference on Applications of Evolutionary Computing*, ser. EuroGP'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 91–102. [En liña]. Dispoñible en: https://doi.org/10.1007/11732242_9

[12] H. Zhou, Y. Zhang, Y. Zhang, and H. Liu, "Feature Selection Based on Conditional Mutual Information: Minimum Conditional Relevance and Minimum Conditional Redundancy," *Applied Intelligence*, vol. 49, no. 3, pp. 883–896, 2019. [En liña]. Dispoñible en: https://doi.org/10.1007/s10489-018-1305-0

[13] H. Peng and Y. Fan, "Feature Selection by Optimizing a Lower Bound of Conditional Mutual Information," *Information Sciences*, vol. 418-419, pp. 652–667, 2017. [En liña]. Dispoñible en: http://www.sciencedirect.com/science/article/pii/S0020025517308836

[14] L. Morán-Fernández, K. Sechidis, V. Bolón-Canedo, A. Alonso-Betanzos, and G. Brown, "Feature Selection with Limited Bit Depth Mutual Information for Portable Embedded Systems," *Knowledge-Based Systems*, vol. 197, p. 105885, 2020. [En liña]. Dispoñible en: http://www.sciencedirect.com/science/article/pii/S0950705120302367

[15] J. González-Domínguez, V. Bolón-Canedo, B. Freire, and J. Touriño, "Parallel Feature Selection for Distributed-Memory Clusters," *Information Sciences*, vol. 496, pp. 399–409, 2019.

[16] O. Soufan, D. Kleftogiannis, P. Kalnis, and V. Bajic, "DWFS: A Wrapper Feature Selection Tool Based on a Parallel Genetic Algorithm," *PLoS ONE*, vol. 10, no. 2, p. e0117988, 2015.

[17] H. Yang, R. Fujimaki, Y. Kusumura, and J. Liu, "Online Feature Selection: A Limited-Memory Substitution Algorithm and Its Asynchronous Parallel Variation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association

for Computing Machinery, 2016, pp. 1945–1954. [En liña]. Dispoñible en: https://doi.org/10.1145/2939672.2939881

[18] J. Yazidi, W. Bouaguel, and N. Essoussi, "A Parallel Implementation of Relief Algorithm Using Mapreduce Paradigm," in *Computational Collective Intelligence*, N. T. Nguyen, L. Iliadis, Y. Manolopoulos, and B. Trawiński, Eds. Cham: Springer International Publishing, 2016, pp. 418–425.

[19] D. Peralta, S. Río, S. Ramírez-Gallego, I. Triguero, J. Benítez, and F. Herrera, "Evolutionary Feature Selection for Big Data Classification: A MapReduce Approach," *Mathematical Problems in Engineering*, vol. 2015, p. 246139, 2015.

[20] Q. He, X. Cheng, F. Zhuang, and Z. Shi, "Parallel Feature Selection Using Positive Approximation Based on MapReduce," in *11th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2014*. IEEE, 2014, pp. 397–402. [En liña]. Dispoñible en: https://doi.org/10.1109/FSKD.2014.6980867

[21] C. Eiras-Franco, V. Bolón-Canedo, S. Ramos, J. González-Domínguez, A. Alonso-Betanzos, and J. Touriño, "Multithreaded and Spark Parallelization of Feature Selection Filters," *Journal of Computational Science*, vol. 17, pp. 609–619, 2016.

[22] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009. [En liña]. Dispoñible en: https://doi.org/10.1145/1656274.1656278

[23] J. González-Domínguez, R. R. Expósito, and V. Bolón-Canedo, "CUDA-JMI: Acceleration of Feature Selection on Heterogeneous Systems," *Future Generation Computer Sciences*, vol. 102, pp. 426–436, 2020.

[24] S. Cuomo, A. Galletti, L. Marcellino, G. Navarra, and G. Toraldo, "On GPU-CUDA as Preprocessing of Fuzzy-Rough Data Reduction by Means of Singular Value Decomposition," *Soft Computing*, vol. 22, pp. 1525–1532, 2017.

[25] H.-H. Chang and C.-Y. Li, "An Automatic Restoration Framework Based on GPU-Accelerated Collateral Filtering in Brain MR Images," *BMC Medical Imaging*, vol. 19, p. 8, 2019.

[26] J. J. Escobar, J. Ortega, J. González, M. Damas, and A. F. Díaz, "Parallel High-Dimensional Multi-Objective Feature Selection for EEG Classification with Dynamic Workload Balancing on CPU–GPU Architectures," *Cluster Computing*, vol. 20, no. 3, pp. 1881–1897, 2017. [En liña]. Dispoñible en: https://doi.org/10.1007/s10586-017-0980-7

[27] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "An Improved GLMNET for l1-Regularized Logistic Regression," *Journal of Machine Learning Research*, vol. 13, pp. 1999–2030, 2012.

[28] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A New Benchmark Collection for Text Categorization Research," *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[29] K. Lang, "Newsweeder: Learning to Filter Netnews," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[31] S. Kogan, D. Levin, B. R. Routledge, J. S. Sagi, and N. A. Smith, "Predicting Risk from Financial Reports with Regression," in *Proceedings of the North American Association for Computational Linguistics Human Language Technologies Conference*, 2009, pp. 272–280.

[32] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, 2011. [En liña]. Dispoñible en: https://doi.org/10.1145/1961189.1961199