



TRABAJO FIN DE GRADO  
INGENIERÍA EN INFORMÁTICA

# undersampling

---

Una biblioteca en Scala para *undersampling* en clasificación no balanceada.

**Autor**

Néstor Rodríguez Vico

**Directores**

Alberto Fernandez Hilario  
Salvador Garcia Lopez



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

---

Granada, 25 de junio de 2018







# **unisersampling**

---

**Una biblioteca en Scala para clasificación no balanceada.**

**Autor**

Néstor Rodríguez Vico

**Directores**

Alberto Fernandez Hilario  
Salvador García Lopez



## **undersampling: una biblioteca en Scala para *undersampling* en clasificación no balanceada.**

Néstor Rodríguez Vico

**Palabras clave:** clasificación no balanceada, preprocesamiento, Scala, reducción de datos.

### **Resumen**

En este trabajo se ha implementado una biblioteca en Scala para solventar el problema del aprendizaje en conjuntos de datos no balanceados. Se han implementado 15 algoritmos de preprocesamiento de datos, los cuales incluyen algoritmos clásicos y algoritmos con un enfoque más modernos. Dichos algoritmos han sido ejecutados sobre más de 20 conjuntos distintos de datos para probar su calidad y poder compararlos posteriormente usando dos clasificadores distintos, un árbol de decisión *C4.5* y una máquina de soporte vectorial.



# **undersampling: a Scala library for *undersampling* in imbalanced classification.**

Néstor Rodríguez Vico

**Keywords:** imbalanced classification, preprocess, Scala, data reduction.

## **Abstract**

In this project, a Scala library has been implemented to solve the learning problem in imbalanced data sets. 15 data preprocessing algorithms have been implemented, including classic algorithms and algorithms with a modern approach. These algorithms have been executed in more than 20 data sets to prove their quality and be compared using two different classifiers, a *C4.5* tree classifier and a support vector machine.



---

Yo, **Néstor Rodríguez Vico**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 75573052C, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Néstor Rodríguez Vico

Granada, 25 de junio de 2018



---

D. **Alberto Fernández Hilario**, Profesor del Área de *Soft Computing and Intelligent Information Systems* del Departamento de Computación y Sistemas Inteligentes de la Universidad de Granada.

D. **Salvador García López**, Profesor del Área de *Soft Computing and Intelligent Information Systems* del Departamento de Computación y Sistemas Inteligentes de la Universidad de Granada.

**Informan:**

Que el presente trabajo, titulado *undersampling, una biblioteca en Scala para undersampling en clasificación no balanceada*, ha sido realizado bajo su supervisión por **Néstor Rodríguez Vico**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 25 de junio de 2018.

**Los directores:**

Alberto Fernández Hilario      Salvador García López



# Agradecimientos

Muchas gracias a mi familia por el apoyo incondicional en todos estos años de carrera, especialmente en este último y duro año. A mis tutores, profesores y amigos que me han ayudado a llegar a donde estoy a día de hoy.

Y a Saray, por estar siempre ahí.



# Índice general

<b>1. Motivación e Introducción</b>	<b>21</b>
1.1. ¿Qué es el aprendizaje automático? . . . . .	21
1.2. ¿Qué es clasificación no balanceada? . . . . .	22
1.3. ¿Tiene solución? . . . . .	22
1.3.1. Técnicas de conjuntos. . . . .	22
1.3.2. Solución a nivel de datos. . . . .	23
1.4. Motivación. . . . .	23
<b>2. Objetivos</b>	<b>25</b>
<b>3. Planificación y Diseño.</b>	<b>27</b>
3.1. Planificación. . . . .	27
3.2. Diseño. . . . .	28
3.3. Presupuesto. . . . .	34
3.4. Requisitos funcionales. . . . .	34
<b>4. Algoritmos seleccionados.</b>	<b>37</b>
4.1. Balance Cascade. . . . .	38
4.2. Class Purity Maximization algorithm. . . . .	38
4.3. ClusterOSS. . . . .	39
4.4. Condensed Nearest Neighbor decision rule. . . . .	40
4.5. Easy Ensemble. . . . .	40
4.6. Edited Nearest Neighbour rule. . . . .	41
4.7. Evolutionary Undersampling. . . . .	41
4.8. Instance Hardness Threshold. . . . .	44
4.9. Iterative Instance Adjustment for Imbalanced Domains. . . . .	45
4.10. NearMiss. . . . .	47
4.11. Neighbourhood Cleaning Rule. . . . .	48
4.12. One-Side Selection. . . . .	49
4.13. Random Undersampling. . . . .	49
4.14. Tomek Link. . . . .	49
4.15. Undersampling Based on Clustering. . . . .	50

<b>5. Introducción a Scala.</b>	<b>53</b>
5.1. ¿Por qué Scala? . . . . .	53
5.1.1. Patrón de diseño mixto. . . . .	53
5.1.2. Simplicidad y elegancia. . . . .	54
5.1.3. Escalabilidad. . . . .	54
5.1.4. Modificadores de visibilidad. . . . .	55
5.1.5. Paralelismo. . . . .	55
5.1.6. Ausencia de Scala. . . . .	55
<b>6. Descripción técnica.</b>	<b>57</b>
6.1. Indices. . . . .	57
6.2. Implementación de los algoritmos. . . . .	58
6.2.1. Logger. . . . .	58
6.2.2. Data. . . . .	58
6.2.3. Algorithm. . . . .	59
6.2.4. Clase BalanceCascade . . . . .	60
6.2.5. Clase ClassPurityMaximization . . . . .	62
6.2.6. Clase ClusterOSS . . . . .	62
6.2.7. Clase CondensedNearestNeighbor . . . . .	63
6.2.8. Clase EasyEnsemble . . . . .	63
6.2.9. Clase EditedNearestNeighbor . . . . .	64
6.2.10. Clase EvolutionaryUnderSampling . . . . .	64
6.2.11. Clase InstanceHardnessThreshold . . . . .	65
6.2.12. Clase IterativeInstanceAdjustmentImbalancedDomains	66
6.2.13. Clase NearMiss . . . . .	67
6.2.14. Clase NeighbourhoodCleaningRule . . . . .	67
6.2.15. Clase OneSideSelection . . . . .	68
6.2.16. Clase RandomUndersampling . . . . .	68
6.2.17. Clase TomekLink . . . . .	68
6.2.18. Clase UndersamplingBasedClustering . . . . .	69
6.3. Funciones de entrada y salida. . . . .	70
6.4. Manual de usuario. . . . .	70
<b>7. Experimentación.</b>	<b>73</b>
7.1. Introducción. . . . .	73
7.2. undersampling . . . . .	73
7.3. Clasificación. . . . .	74
7.3.1. Porcentaje de acierto. . . . .	75
7.3.2. AUC. . . . .	75
7.3.3. Media geométrica. . . . .	77
7.4. Resultados numéricos. . . . .	77
7.5. Gráficos. . . . .	87

---

**ÍNDICE GENERAL** **15**

<b>8. Conclusiones y trabajos futuros.</b>	<b>89</b>
8.1. Trabajos futuros. . . . .	89
<b>9. Apéndice.</b>	<b>91</b>
<b>Bibliografía</b>	<b>91</b>



# Índice de tablas

7.1.	Tamaño de los conjuntos de datos reducidos. . . . .	79
7.2.	Porcentaje de reducción. . . . .	80
7.3.	<i>Imbalance Ratio</i> de los conjuntos de datos reducidos. . . . .	81
7.4.	Métricas clasificacion (parte 1). . . . .	82
7.5.	Métricas clasificacion (parte 2). . . . .	83
7.6.	Métricas clasificacion (parte 3). . . . .	84
7.7.	Métricas clasificacion (parte 4). . . . .	85
7.8.	Tiempos de ejecución. . . . .	86
9.1.	Métricas usando k-fold de los conjuntos originales. . . . .	92
9.2.	Métricas usando k-fold de los conjuntos procesados (parte 1). . . . .	93
9.3.	Métricas usando k-fold de los conjuntos procesados (parte 2). . . . .	94
9.4.	Métricas usando k-fold de los conjuntos procesados (parte 3). . . . .	95
9.5.	Métricas usando k-fold de los conjuntos procesados (parte 4). . . . .	96
9.6.	Métricas usando k-fold de los conjuntos procesados (parte 5). . . . .	97
9.7.	Métricas usando k-fold de los conjuntos procesados (parte 6). . . . .	98
9.8.	Métricas usando k-fold de los conjuntos procesados (parte 7). . . . .	99
9.9.	Métricas usando k-fold de los conjuntos procesados (parte 8). . . . .	100
9.10.	Métricas usando k-fold de los conjuntos procesados (parte 9). . . . .	101
9.11.	Métricas usando k-fold de los conjuntos procesados (parte 10). . . . .	102
9.12.	Métricas usando k-fold de los conjuntos procesados (parte 11). . . . .	103



# Índice de figuras

3.1. Planificación del proyecto . . . . .	27
7.1. Ejemplo curva ROC. . . . .	76
7.2. Nubes de puntos para <i>banana</i> . . . . .	88



# Capítulo 1

## Motivación e Introducción

Vivimos en una era de datos, una era en la que generamos miles de millones de datos cada día y una era en la que nos esforzamos por aprovechar dichos datos para nuestro beneficio. Es en el interés por aprovechar los datos que tenemos donde surge el *Machine Learning*.

### 1.1. ¿Qué es el aprendizaje automático?

El aprendizaje automático (del inglés, “Machine Learning”) [1] es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a los ordenadores aprender. La idea es crear programas que sean capaces de abstraer las características y comportamientos a partir de una conjunto de datos proporcionados como ejemplos. Una vez dichos programas han aprendido, podemos usarlos para predecir el comportamiento de datos futuros de la misma fuente. El concepto predecir es ambiguo, pero podemos dividir los problemas en dos grupos según el concepto de predecir: problemas de clasificación y problemas de regresión.

- **Clasificación.** En los problemas de clasificación se intenta predecir la clase de las instancias —una instancia es un dato—. La clase de una instancia se podría definir como el grupo al que pertenece una instancia. Un ejemplo clásico de un problema de clasificación es decidir si en una fotografía aparece una persona o no. Tendríamos un programa que, dado un conjunto sustancioso de fotos —algunas con personas y otras sin personas para poder aprender correctamente—, es capaz de extraer las características que le permitirán determinar en una foto nueva si aparece una persona o no. En este caso, tenemos dos posibles clases, *persona* y *no persona*. Este ejemplo es un problema de clasificación binaria, es decir, solo tenemos dos clases. En la vida real existen problemas de clasificación multiclasa, es decir, problemas en los que las clases a predecir son más de dos.

- **Regresión.** A diferencia de los problemas de clasificación, los elementos del conjunto de datos no están etiquetados por una clase como tal, sino por ciertos valores. A la hora de predecir en los problemas de regresión tratamos de acercarnos lo máximo posible al valor real del dato con el que estamos trabajando. Un ejemplo clásico de un problema de regresión es el de predecir el valor de un objeto. En este caso no podemos decidir entre  $N$  posibles precios —las que serían las  $N$  clases en un problema de clasificación—, sino que debemos predecir un valor que sea totalmente nuevo para nuestro problema.

Este proyecto se centra en resolver problemas de clasificación binaria, es decir, conjuntos de datos en los que cada dato pertenece a una clase u otra.

## 1.2. ¿Qué es clasificación no balanceada?

Cuando se intenta aprender de un conjunto de datos, el caso idílico es aquel en el que la distribución del número de instancias asociado a cada clase del problema es uniforme, es decir, tenemos un número similar de instancias para cada clase. Pero esto no siempre es así. Por ejemplo, pensemos en resolver el problema de clasificar una transacción bancaria como fraudulenta o no. Para resolver este problema, podemos pensar en resolverlo aplicando *Machine Learning*. Para ello podemos pedirle a todos los bancos del mundo un listado de sus transacciones bancarias y la clase asociada a cada una de ellas: *fraudulenta* o *legal*. Si hacemos esto, seguramente tengamos un conjunto de datos bastante amplio. El problema reside en el número de transacciones fraudulentas existentes por cada número de transacciones legales. Este conjunto de datos tendrá muchas instancias asociadas a la clase *legal* y muy pocas instancias asociadas a la clase *fraudulenta*. Si en el problema al que nos enfrentamos la clase de interés —en este caso la clase *fraudulenta*— tiene un número bastante menor de instancias que el resto de clase nos encontramos ante un problema de clasificación no balanceada.

## 1.3. ¿Tiene solución?

### 1.3.1. Técnicas de conjuntos.

Esta vertiente busca modificar los algoritmos de clasificación existentes para adaptarlos a conjuntos de datos no balanceados. El objetivo principal de esta metodología es mejorar el rendimiento de los clasificadores. Para ello, se construyen varios subconjuntos de datos sobre los cuales se aprende un clasificador para cada uno de ellos y, finalmente, se combina el resultado de cada partición. En esta biblioteca se incluyen algoritmos que usan esta vertiente.

### 1.3.2. Solución a nivel de datos.

Este tipo de técnicas buscan equilibrar las clases en los datos de entrenamiento (preprocesamiento de datos) antes de proporcionar los datos al algoritmo de aprendizaje automático. El objetivo principal de equilibrar clases es aumentar la frecuencia de la clase minoritaria —técnica conocida como *oversampling*— o disminuir la frecuencia de la clase mayoritaria —técnica conocida como *undersampling*—. Ambas vertientes buscan obtener un conjunto de datos donde existe aproximadamente el mismo número de instancias para ambas clases.

En este proyecto se ha optado por esta vertiente, en concreto por algoritmos de *undersampling*. Hay que matizar que la mayoría de ellos busca reducir el número de instancias de la clase mayoritaria, pero también puede darse el caso de que alguno de ellos elimine también instancias minoritarias con idea de reducir el ruido o la redundancia de datos.

## 1.4. Motivación.

Este proyecto surge debido a la necesidad de tener en una única biblioteca de software libre y código abierto —todo el código está disponible en mí GitHub [2] y la documentación se encuentra en <https://nestorrv.github.io>— con la mayor cantidad de algoritmos de *undersampling* posibles. También se ha hecho en un lenguaje de programación el cual está al alza y en el cuál no hay una biblioteca tan completa como esta para aprendizaje no balanceado.



## Capítulo 2

# Objetivos

En este capítulo se detallan los objetivos del proyecto.

- Revisión bibliográfica del estado del arte. Este objetivo pretende explorar los algoritmos ya publicados, para adquirir un conocimiento fuerte que permita implementar una biblioteca lo más completa posible.
- Estudio de requisitos y diseño de implementación. Dado que se trata de un proyecto grande, se ha de realizar un estudio de requisitos y realizar un diseño correcto para no tener que reestructurar más de la cuenta el código.
- Implementación y prueba de los algoritmos. Aquí se implementaran los quince algoritmos seleccionados y se comprobará su correcto funcionamiento ejecutándolos contra una gran batería de conjuntos de prueba así como comparando el resultado con los resultados obtenidos usando la biblioteca *imbalanced-learn* [3].
- Comparación de los resultados obtenidos. Dado que hay múltiples algoritmos, la idea es realizar una comparación de los resultados obtenidos tras aplicar los algoritmos y realizar otra comparación obteniendo distintas métricas al usado dos clasificadores distintos, un árbol de decisión *C4.5* [4] y una *SVM* [5] —acrónimo de *Support Vector Machine*, *máquina de soporte vectorial* en español—.



## Capítulo 3

# Planificación y Diseño.

### 3.1. Planificación.

En un proyecto de estas dimensiones la planificación es fundamental. La planificación que he seguido la podemos ver en el siguiente diagrama de Gantt:

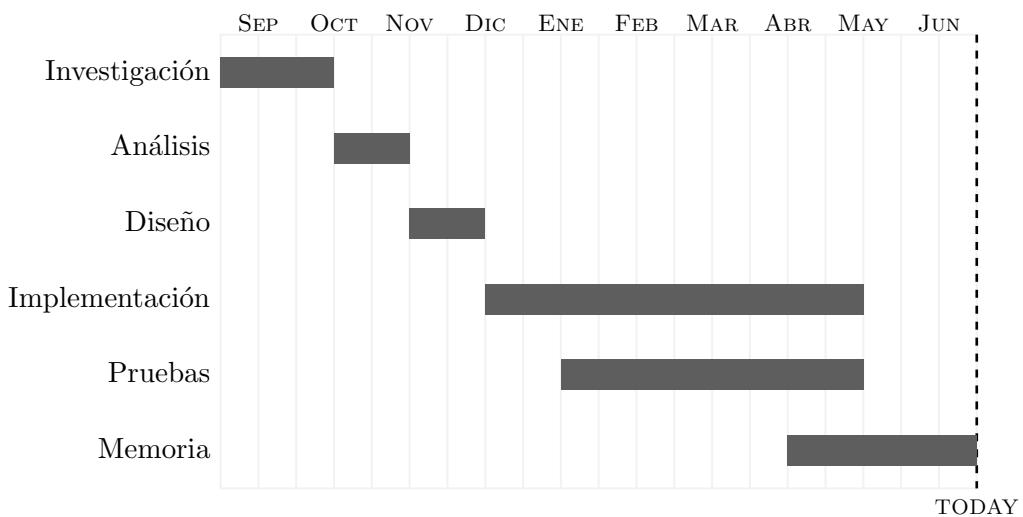


Figura 3.1: Planificación del proyecto

La planificación era bastante directa; las primeras semanas de investigación, para adquirir el conocimiento necesario para poder desarrollar este proyecto. A continuación, dado que es una biblioteca de gran tamaño, un mes de trabajo desarrollando diagramas de clases y analizando las mejores opciones para implementar los algoritmos. Finalmente, la parte más dura, implementar los algoritmos. Primero empecé implementando las estructuras

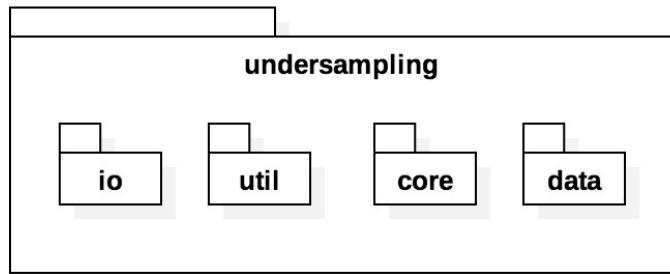
de datos y funciones auxiliares que iba a necesitar para posteriormente facilitar la implementación de los algoritmos en sí. Desde que implementé el primero, comenzaron las pruebas para ver que los algoritmos eran correctos. Cuando se acercaba la etapa final del proyecto comencé con la memoria para no quedarme sin tiempo.

### 3.2. Diseño.

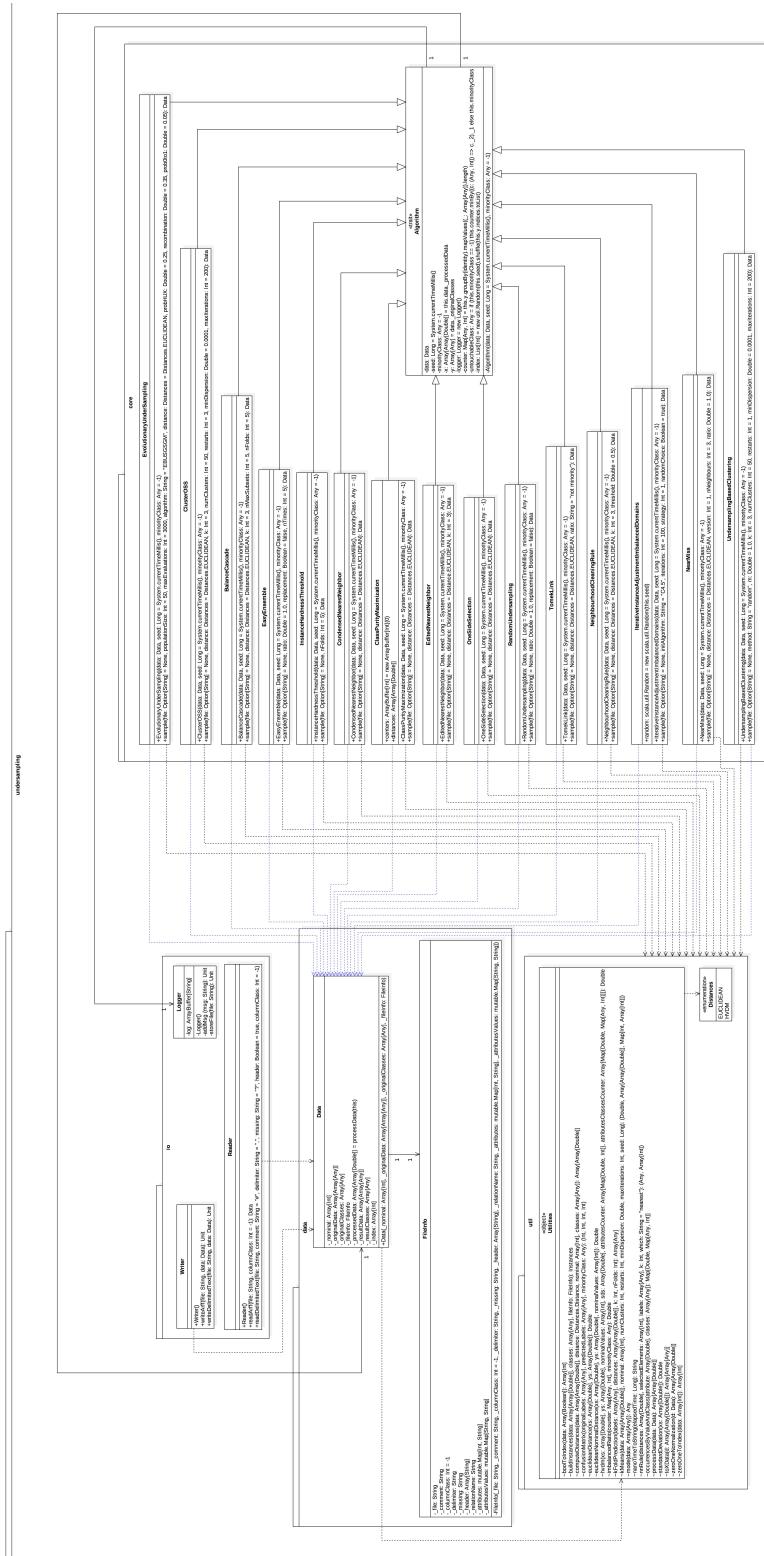
Para poder agrupar las clases diseñadas según su funcionalidad se han creado los siguientes paquetes:

- *core*: paquete principal de la biblioteca. Se usa para contener las clases asociadas a todos los algoritmos implementados.
- *data*: paquete usado para contener todas las estructuras de datos e información usada por los algoritmos.
- *io*: paquete usado para contener todas las clases destinadas a entrada y salida de datos.
- *util*: paquete usado para contener una clase de utilidades, es decir, funciones usadas por varios algoritmos.

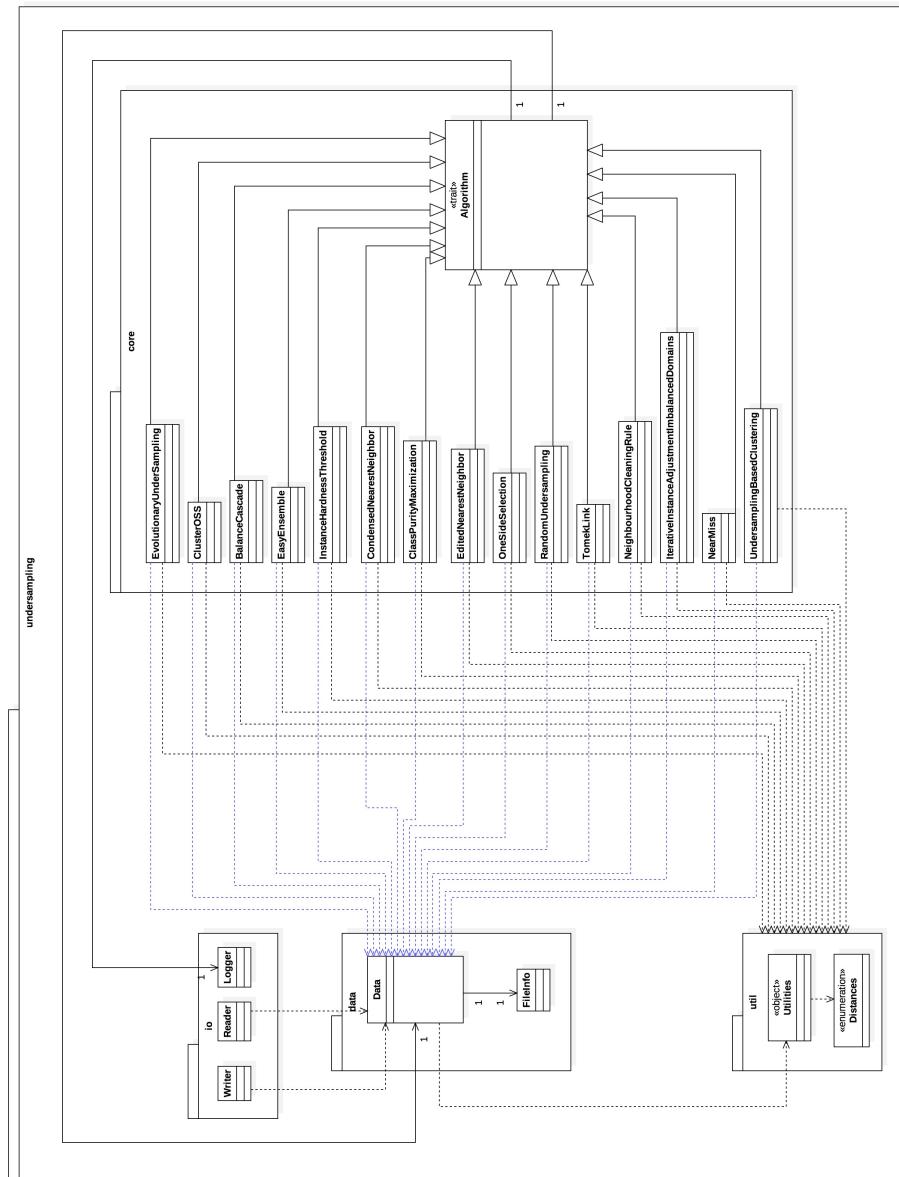
Todos los paquetes nombrados anteriormente se ubican dentro del paquete *undersampling*. El diagrama de paquetes asociado a la biblioteca lo podemos ver a continuación:



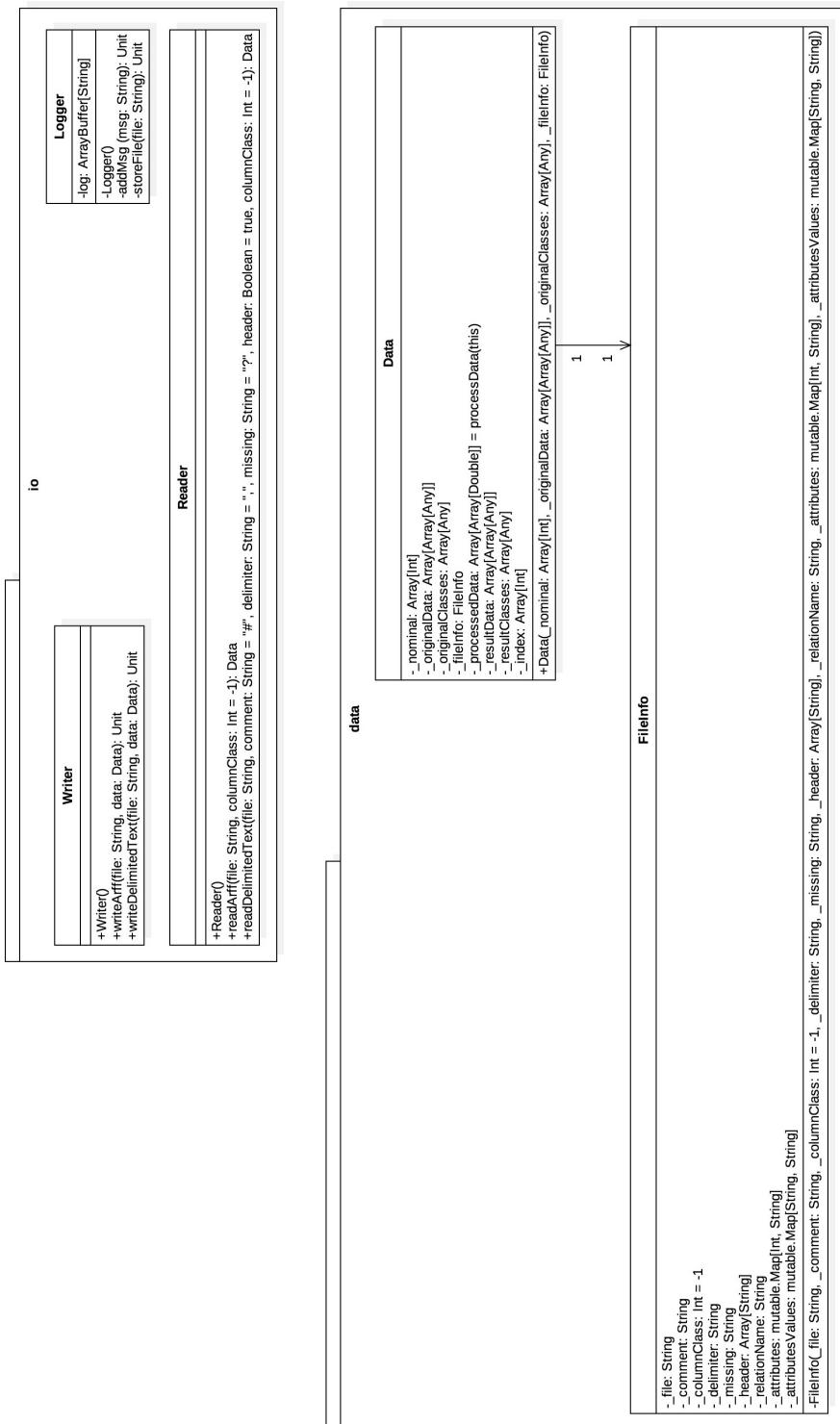
Tras diseñar los paquetes a usar, diseñé las clases que iba a necesitar para el proyecto. La estructura es bastante sencilla. Los algoritmos están implementados cada uno en su clase y todas ellas heredan de *Algorithm*. Los algoritmos usan algunas funciones definidas en la clase *Utilities* y usan un valor del enumerado *Distances* para indicar la distancia a usar. Todos los algoritmos hacen uso de la estructura de datos representada por la clase *Data*. A continuación podemos ver el diagrama de clases implementado:

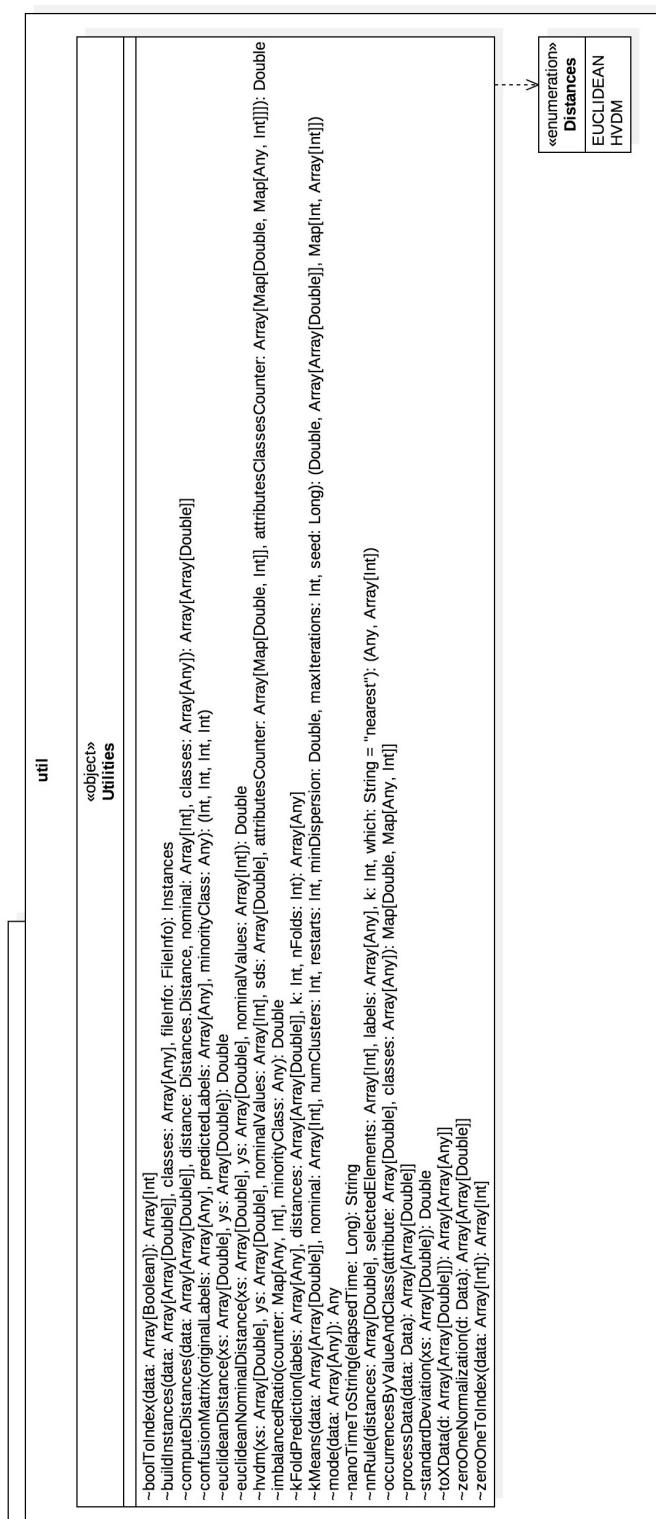


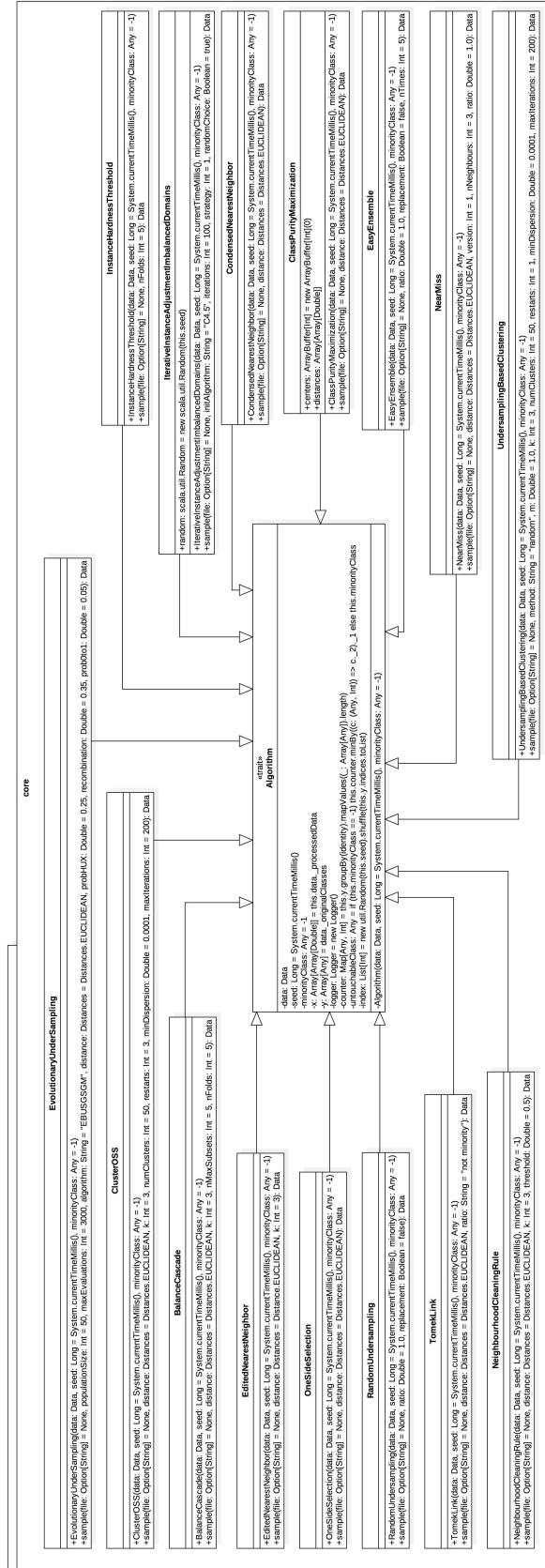
Dado que no se puede ver con claridad cada clase, en la siguiente figura podemos ver un diagrama de clases simplificado, es decir, sin los métodos ni atributos de cada clase:



Para poder apreciar los detalles de cada clase, podemos ver las siguientes figuras:







### 3.3. Presupuesto.

Para la implementación de este proyecto se hubiese necesitado un presupuesto más o menos básico:

- *Un ordenador para desarrollo:* en mi caso se ha utilizado un *MacBook Pro* del fabricante *Apple* valorado en *3300 euros*. Este es mi ordenador personal y por eso lo he usado, pero no sería necesario un ordenador tan potente. Un ordenador con un sistema *Unix* y un procesador medianamente en condiciones hubiese sido suficiente, reduciendo el coste a unos *800-1000 euros*.
- *Conexión a Internet:* recurso indispensable para acceder a la documentación de *Scala* y los *papers* de los artículos. Dado que no se hacen grandes transferencias de datos, tampoco es necesario una conexión de gran velocidad. En mi caso se ha usado una conexión de fibra óptima de *50Mb* con un precio de unos *30 euros* al mes.
- *Suministro eléctrico:* necesario para poder alimentar el ordenador. Dado que se trata de una biblioteca que maneja grande volúmenes de datos se hace un uso intenso del procesador, lo cual supone un alto consumo de batería y por tanto eléctrico.
- *Salario:* estimo haber echado unas 350 horas en realizar todo el proyecto, incluyendo todas las partes del mismo. Supongamos que se pagan a diez euros la hora, el dinero necesario rondaría los tres mil quinientos euros.

### 3.4. Requisitos funcionales.

Para que se trate de una biblioteca competitiva debe cumplir los siguiente requisitos:

- *Alta velocidad de procesamiento:* se trata de una biblioteca que puede manejar grandes volúmenes de datos. Si para conjuntos pequeños el tiempo de ejecución es elevado, este será enorme cuando se introduzca mayor volumen de datos.
- *Diferentes formatos de entrada:* hoy en día los datos pueden venir de distintos origines y, por lo tanto, debemos estar preparados para leer una gran variedad de formatos. Esta biblioteca es capaz de leer datos en formato *ARFF* [6] de *Weka* [7] [8] y datos en cualquier formato de texto plano indicando los tokens necesarios, como puede ser el separados de elementos, el token de comentario, el token que representa valores nulos, *etcetera*.

- *Diferentes formatos de salida:* al igual que sucede con la entrada de datos, sucede con los datos de salida. Esta biblioteca permite los mismos formatos de salida que de entrada, facilitando así al usuario la posibilidad de tener el resultado de los algoritmos en el mismo formato de entrada —u otro a elegir—.
- *Seguro a errores:* este tipo de algoritmos tiene una gran variedad de parámetros y, por lo tanto, distintas configuraciones que el usuario puede elegir. Todos los algoritmos están probados con distintos valores de estos parámetros y los propios algoritmos comprueban que los argumentos son válidos.
- *Datos intactos:* los datos de entrada son modificados en la mayoría de los algoritmos para poder trabajar con ellos, pero no nos podemos permitir devolver los datos modificados, ya que esos no son los datos que ha introducido el usuario. Para evitar esto, todos los algoritmos —con excepción del algoritmo *IPADE-ID* ya que crea nuevos datos sintéticos y, por lo tanto, no es posible garantizar esta condición— trabajan con índices a los datos en vez de con los propios datos, para así no poder devolver los datos originales accediendo a las posiciones indicadas por el índice. Esto también nos proporciona una mayor velocidad de procesamiento, ya que es más eficiente trabajar con una colección de índices que con una matriz de  $n \times m$  elementos.



## Capítulo 4

# Algoritmos seleccionados.

En este proyecto se han implementado los siguientes 15 algoritmos:

- **Balance Cascade**, al cual nos referiremos como *BC*.
- **Class Purity Maximization algorithm**, al cual nos referiremos como *CPM*.
- **ClusterOSS**, al cual nos referiremos como *ClusterOSS*.
- **Condensed Nearest Neighbor decision rule**, al cual nos referiremos como *CNN*.
- **Easy Ensemble**, al cual nos referiremos como *EE*.
- **Edited Nearest Neighbour rule**, al cual nos referiremos como *ENN*.
- **Evolutionary Undersampling**, al cual nos referiremos como *EUS*.
- **Instance Hardness Threshold**, al cual nos referiremos como *IHTS*.
- **Iterative Instance Adjustment for Imbalanced Domains**, al cual nos referiremos como *IPADE-ID*.
- **NearMiss**, al cual nos referiremos como *NM*.
- **Neighbourhood Cleaning Rule**, al cual nos referiremos como *NCL*.
- **One-Side Selection**, al cual nos referiremos como *OSS*.
- **Random Undersampling**, al cual nos referiremos como *RU*.
- **Tomek Link**, al cual nos referiremos como *TL*.
- **Undersampling Based on Clustering**, al cual nos referiremos como *SCB*.

En este capítulo vamos a ver el pseudocódigo de cada uno de ellos y explicar un poco su funcionamiento e idea principal.

### 4.1. Balance Cascade.

El artículo original donde se ha publicado este algoritmo es “*Exploratory Undersampling for Class-Imbalance Learning*” escrito por *Xu-Ying Liu, Jianxin Wu y Zhi-Hua Zhou* [9]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function BALANCE CASCADE(Conjunto de instancias minoritaria P,
   conjuntos de instancias mayoritarias N con  $|P| < |N|$ , numero de sub-
   conjuntos T a crear de N, número de iteraciones de AdaBoost  $s_i$ )
2:    $i \leftarrow 0, f \leftarrow \sqrt[T-1]{\frac{|P|}{|N|}}$ , f es el ratio de falsos positivos que  $H_i$  debe
   alcanzar
3:   while  $i \neq T$  do
4:      $i \leftarrow i + 1$ 
5:     Coger de forma aleatoria un subconjunto  $N_i$  con  $|P| < |N_i|$ 
6:     Aprender  $H_i$  con P,  $N_i$  donde  $H_i$  es un clasificador AdaBoost
       con  $s_i$  clasificadores debiles  $h_{i,j}$  y correspondientes pesos  $\alpha_{i,j}$ . El
       umbral es  $\theta_i$ .  $H_i = sgn(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i)$ 
7:     Ajustar  $\theta_i$  para que el ratio de falsos positivos de  $H_i$  sea f.
8:     Quitar de N todos los ejemplos que son clasificados correctamen-
       te por  $H_i$ 
9:   end while
10:  return Un conjunto:  $H(x) = sgn(\sum_{i=1}^T \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^T \theta_i)$ 
11: end function
```

---

La idea de este algoritmo es realizar distintas particiones del conjunto de datos inicial y aprender un clasificador para cada uno de ellos y luego combinar las mejores instancias de cada subconjunto para formar el conjunto final de datos.

### 4.2. Class Purity Maximization algorithm.

El artículo original donde se ha publicado este algoritmo es “*An Unsupervised Learning Approach to Resolving the Data Imbalanced Issue in Supervised Learning Problems in Functional Genomics*” escrito por *Kihoon Yoon y Stephen Kwek* [10]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function CLASSPURITYMAXIMIZATION
2:   Seleccionar como centros una instancia de la clase minoritaria y
      otra de la clase mayoritaria
```

---

- 
- 3: Repartir el resto de instancias en cada uno de los *clúster* generados en el paso dos según su cercanía a cada uno de los centros.  
Se debe garantizar que uno de los dos *clúster* tiene un mayor valor de pureza  
4: Repetir el proceso recursivamente —en cada uno de los *clúster* generados en el paso anterior— hasta que no se pueda generar una partición en la cual uno de los *subclúster* tenga mayor pureza que su *clúster* padre  
5: **return** Añadimos todas las instancias minoritarias a cada *clúster* no puro  
6: **end function**
- 

La idea de este algoritmo es ir partiendo el conjunto de datos en *clústeres* e ir realizando el particionamiento mientras obtengamos un conjunto más puro —mejor— que el anterior.

### 4.3. ClusterOSS.

El artículo original donde se ha publicado este algoritmo es “*ClusterOSS: a new undersampling method for imbalanced learning.*” escrito por *Victor H Barella, Eduardo P Costa y André C. P. L. F. Carvalho* [11]. El pseudocódigo del algoritmo es el siguiente:

- 
- 1: **function** CLUSTER OSS(Conjunto de datos D)  
2: Train = {} Test = {}  
3: InstanciasMayoritarias = cogerInstanciasMayoritarias(D)  
4: C = AlgClustering(InstanciasMayoritarias)  
5: **for** cluster  $C_i \in C$  **do**  
6:     x = InstanciaMasCercanaAlCentro( $C_i$ )  
7:     Train = Train  $\cup \{x\}$   
8:     Test = Test  $\cup (C_i \setminus \{x\})$   
9: **end for**  
10: Resultado = KNN(Train, Test)  
11: MalClasificados = ObtenerMalClasificados(Resultado)  
12: NuevoD = Train  $\cup$  MalClasificados  
13: TLinks = TomekLinks(NuevoD)  
14: **for** z  $\in$  NuevoD **do**  
15:     **if** z  $\in$  TLinks **then**  
16:         NuevoD = NuevoD - {z}  
17:     **end if**  
18: **end for**  
19: **return** NuevoD  
20: **end function**
- 

*AlgClustering* es una función que aplica un algoritmo de *clustering*. En

este proyecto se ha usado *kMeans* [12].

La idea de este algoritmo es construir varios *clústeres* para quedarnos con los elementos más representativos del conjunto de datos original. Una vez tenemos los elementos más representativos, aplicamos un algoritmo *Tomek Link* —el cual podemos ver en la sección 4.14— para limpiar elementos ruidosos.

#### 4.4. Condensed Nearest Neighbor decision rule.

El artículo original donde se ha publicado este algoritmo es “*The Condensed Nearest Neighbor Rule*” escrito por *P. Hart* [13]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function CNN
2:   El primer ejemplo se coloca en STORE
3:   El segundo ejemplo se clasifica usando la regla NN usando como
      conjunto de datos el contenido de STORE. Si se clasifica correcta-
      mente, dicho ejemplo se mete en GRABBAG, en caso contrario
      se pone en STORE
4:   De forma iterativa, se repite el proceso del paso 2 para cada
      ejemplo restante
5:   Tras una primera pasada por los datos, se procede a iterar sobre
      GRABBAG hasta terminar, lo cual puede suceder por:
      ▪ Todos los elementos de GRABBAG se han transferido a STORE.
        En este caso, el conjunto obtenido es igual que el original
      ▪ Se ha realizado una pasada entera sobre GRABBAG y ningún
        elemento ha sido transferido a STORE
6:   return El contenido de STORE es el resultado final
7: end function
```

---

La idea de este algoritmo es bien sencilla. Busca eliminar puntos innecesarios. Un punto se considera innecesario si con el conjunto de puntos que llevamos guardados hasta el momento somos capaces de predecir correctamente su clase. Si esto es así, no nos interesa ese punto y lo descartamos.

#### 4.5. Easy Ensemble.

El artículo original donde se ha publicado este algoritmo es “*Exploratory Undersampling for Class-Imbalance Learning*” escrito por *Xu-Ying Liu, Jianxin Wu y Zhi-Hua Zhou* [9]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function EASYENSEMBLE(Conjunto de instancias minoritaria P, conjuntos de instancias mayoritarias N con  $|P| < |N|$ , numero de subconjuntos T a crear de N, número de iteraciones de AdaBoost  $s_i$ )
2:    $i \leftarrow 0$ 
3:   while  $i \neq T$  do
4:      $i \leftarrow i + 1$ 
5:     Coger de forma aleatoria un subconjunto  $N_i$  con  $|P| < |N_i|$ 
6:     Aprender  $H_i$  con P,  $N_i$  donde  $H_i$  es un clasificador AdaBoost con  $s_i$  clasificadores debiles  $h_{i,j}$  y correspondientes pesos  $\alpha_{i,j}$ . El umbral es  $\theta_i$ .  $H_i = \text{sgn}(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i)$ 
7:   end while
8:   return Un conjunto:  $H(x) = \text{sgn}(\sum_{i=1}^T \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^T \theta_i)$ 
9: end function

```

---

Este algoritmo sigue la misma idea que *BC* 4.1. Busca realizar distintas particiones del conjunto de datos inicial y aprender un clasificador para cada uno de ellos y luego combinar las mejores instancias de cada subconjunto para formar el conjunto final de datos.

#### 4.6. Edited Nearest Neighbour rule.

El artículo original donde se ha publicado este algoritmo es “*Asymptotic Properties of Nearest Neighbor Rules Using Edited Data*” escrito por *Dennis L. Wilson* [14]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function ENN(Conjunto de datos D)
2:   S = D
3:   for  $x_i \in S$  do
4:     Descartamos  $x_i$  de S si no se clasifica correctamente usando la regla NN usando como conjunto de datos  $D - \{x_i\}$ 
5:   end for
6:   return S
7: end function

```

---

La idea de este algoritmo es bastante similar a la de *CNN* 4.4, va eliminando puntos del conjunto de datos si se consideran redundantes, es decir, podemos predecir su clase usando todos los puntos del conjunto original quitando el que estamos tratando.

#### 4.7. Evolutionary Undersampling.

El artículo original donde se ha publicado este algoritmo es “*Evolutionary Under-Sampling for Classification with Imbalanced Data Sets: Proposals y Taxonomy*” escrito por *Salvador Garcia y Francisco Herrera* [15]. El

pseudocódigo de este algoritmo es el pseudocódigo de cualquier algoritmo evolutivo, el cuál es el siguiente:

---

```

1: function EUS(Conjunto de datos D)
2:   Generar población aleatoria inicial
3:   while numeroEvaluaciones < evaluacionesMaximas do
4:     Generamos los nuevos descendientes a partir de la población
       actual
5:     Evaluamos los nuevos descendientes
6:     Ordenamos la población actual y la nueva población
7:     Mezclamos las dos poblaciones quedándonos con los N mejores
       individuos, donde N es el tamaño original de la población.
8:   end while
9:   return El mejor individuo de la población
10: end function

```

---

Dado que es un algoritmo evolutivo, podemos describir las características principales del mismo:

- *Representación del individuo*: cada individuo es un vector de tamaño  $|conjunto\_original|$  formado por unos y/o ceros. Cada individuo representa un subconjunto de elementos del conjunto original. Si en la posición  $i$  del vector hay un uno, la instancia  $i$  está incluida en dicho subconjunto.
- *Población inicial*: población aleatoria formada por el número de individuos indicado por el usuario.
- *Función de evaluación*: este algoritmo admite distintas funciones de evaluación según la versión deseada a ejecutar.
  - Para la versión *EBUSGSGM* se usa

$$g - \text{abs}(1 - (nPositives/nNegatives)) * 20 \quad (4.1)$$

donde  $g$  es la media geométrica —la cual podemos ver explicada con detalle en la subsección 7.3.3—,  $nPositives$  es el número de instancias positivas en el subconjunto seleccionado y  $nNegatives$  es el número de instancias negativas en el subconjunto seleccionado.

- Para la versión *EBUSMSGM* se usa

$$g - \text{abs}(1 - (nPositivesTotal/nNegatives)) * 20 \quad (4.2)$$

donde  $g$  es la media geométrica,  $nPositivesTotal$  es el número de instancias positivas en el conjunto original y  $nNegatives$  es el número de instancias negativas en el subconjunto seleccionado.

- Para la versión *EUSCMGSGM* se usa la media geométrica.
- Para la versión *EUSCMMMSGM* se usa la media geométrica.
- Para la versión *EBUSGSAUC* se usa

$$auc - abs(1 - (nPositives/nNegatives)) * 0.2 \quad (4.3)$$

donde *auc* es el área bajo la curva *ROC* —la cual podemos ver explicada con detalle en la subsección 7.3.2—, *nPositives* es el número de instancias positivas en el subconjunto seleccionado y *nNegatives* es el número de instancias negativas en el subconjunto seleccionado.

- Para la versión *EBUSMSAUC* se usa

$$auc - abs(1 - (nPositivesTotal/nNegatives)) * 0.2 \quad (4.4)$$

donde *auc* es el área bajo la curva *ROC* *nPositivesTotal* es el número de instancias positivas en el conjunto original y *nNegatives* es el número de instancias negativas en el subconjunto seleccionado.

- Para la versión *EUSCMGSAUC* se usa el área bajo la curva *ROC*.
  - Para la versión *EUSCMMSAUC* se usa el área bajo la curva *ROC*.
- *Criterio de parada:* el algoritmo acepta un parámetro mediante el cual el usuario puede indicar el número de evaluaciones límite que puede hacer el algoritmo.
  - *Operador de cruce:* se ha usado el operador de cruce *HUX*. Este operador genera dos nuevos descendientes intercambiando los genes —un gen es un elemento del vector que representa al individuo— de sus dos ancestros. El intercambio se produce si la distancia *Hamming* —el número de genes distintos entre los dos ancestros— es mayor que un umbral indicado por el usuario.

Las versiones cuyo nombre contiene *GS* indican que se aplica *Global Selection*, es decir, se pueden descartar instancias de la clase minoritaria. Las versiones cuyo nombre contiene *MS* indican que se aplica *Majority Selection*, es decir, sólo se pueden descartar instancias de la clase mayoritaria.

Las versiones cuyo nombre contiene *GM* indican que se usa la media geométrica como componente principal de la función de evaluación. Las versiones cuyo nombre contiene *AUC* indican que se usa el área bajo la curva *ROC* como componente principal de la función de evaluación.

Las versiones cuyo nombre contiene *EBUS* —que significa *Evolutionary Balancing Under-Sampling*— priorizan obtener un conjunto de datos equilibrado. Las versiones cuyo nombre contiene *EUSCM* —que significa *Evolutionary Under-Sampling guided by Classification Measures*— priorizan obtener un conjunto de datos con buenos resultados a la hora de realizar clasificación, dejando en un segundo plano el objetivo de buscar un conjunto de datos equilibrado.

## 4.8. Instance Hardness Threshold.

El artículo original donde se ha publicado este algoritmo es “*An Empirical Study of Instance Hardness*” escrito por *Michael R. Smith, Tony Martinez y Christophe Giraud-Carrier* [16]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function IHTS(Conjunto de datos D)
2:   Calcular  $k - DisagreeingNeighbors$  –  $kDN$ 
3:   Calcular  $DisjunctSize$  –  $DS$ 
4:   Calcular  $DisjunctClassPercentage$  –  $DCP$ 
5:   Calcular  $ClassLikelihood$  –  $Cl$ 
6:   Calcular  $ClassLikelihoodDifference$  –  $CLD$ 
7:   Calcular  $MinorityValue$  –  $MV$ 
8:   Calcular  $ClassBalance$  –  $CB$ 
9:   La dureza de una instancia se calcula como  $0.5569 * DN - 0.1984 * DCP - 0.124 * CL + 0.0752 * CB - 0.072 * CLD + 0.0365 * DS + 0.0339 * MV + 0.9088$ 
10:  Clasificar las instancias según su dureza: alta si  $(CLD(x, t(x)) < 0 \text{ y } (DS(x) == 0 \text{ y } DCP(x) < 0.5) \text{ o } DN(x) > 0.8)$ , baja si  $((DS(x) == 0 \text{ y } DCP(x) < 1) \text{ o } DN(x) > 0.2)$  o ninguna en otro caso.
11:  Las instancias con una dureza alta se consideran que están mal clasificadas o son ruidosas y, por lo tanto, son eliminadas del conjunto de datos.
12: end function
```

---

$kDN$  se puede calcular de la siguiente forma:

$$kDN(x) = \frac{\{y : y \in kNN(x) \wedge t(y) \neq t(x)\}}{k} \quad (4.5)$$

donde  $kNN(x)$  es el conjunto de los  $k$  vecinos más cercanos de  $x$  y  $t(x)$  es la clase asociada a  $x$ .

$DS$  se puede calcular de la siguiente forma:

$$DS(x) = \frac{|\text{disjunct}(x) - 1|}{\max_{y \in D} |\text{disjunct}(y) - 1|} \quad (4.6)$$

donde  $\text{disjunct}(x)$  es una función que devuelve el subconjunto de un árbol de decisión  $C4.5$  que cubre a  $x$ .

DCP se puede calcular de la siguiente forma:

$$DCP(x) = \frac{|\{z : z \in \text{disjunct}(x) \wedge t(z) = t(x)\}|}{|\text{disjunct}(x)|} \quad (4.7)$$

donde  $\text{disjunct}(x)$  es una función que devuelve el subconjunto de un árbol de decisión  $C4.5$  que cubre a  $x$  y  $t(x)$  es la clase asociada a  $x$ .

CL se puede calcular de la siguiente forma:

$$CL(x, t(x)) = \prod_i^{|x|} P(x_i | t(x)) \quad (4.8)$$

donde  $x_i$  es el valor de la instancia  $x$  en su atributo  $i$ -ésimo.

CLD es la diferencia entre CL y la máxima probabilidad para todas las otras clases. Se puede calcular de la siguiente forma:

$$CLD(x, t(x)) = CL(x, t(x)) - \underset{y \in Y - t(x)}{\operatorname{argmax}} CL(x, y) \quad (4.9)$$

MV se puede calcular de la siguiente forma:

$$MV(x) = \frac{|\{z : z \in D \wedge t(z) = t(x)\}|}{\max_{y \in Y} |\{z : z \in D \wedge t(z) = y\}|} \quad (4.10)$$

Finalmente, CB se puede calcular de la siguiente forma:

$$MV(x) = \frac{|\{z : z \in D \wedge t(z) = t(x)\}|}{|D|} - \frac{1}{|Y|} \quad (4.11)$$

## 4.9. Iterative Instance Adjustment for Imbalanced Domains.

El artículo original donde se ha publicado este algoritmo es “*Addressing imbalanced classification with instance generation techniques: IPADE-ID*” escrito por *Victoria López, Isaac Triguero, Cristóbal J. Carmona, Salvador García y Francisco Herrera* [17]. El pseudocódigo del algoritmo es el siguiente:

#### 46 4.9. Iterative Instance Adjustment for Imbalanced Domains.

---

```
1: function IPADE-ID(Conjunto de datos D)
2:   GS = Inicializar la población usando un árbol de decisión C4.5
      o kNN
3:   Mejorar la población GS aplicando un algoritmo de Evolución
      Diferencial
4:   AUC = evaluar población GS con TR
5:   registro de clases[0..nClases] = optimizable
6:   numOptim[0..nClases] = 0
7:   while AUC != 1 quedan clases optimizables do
8:     menorPrecision =  $\infty$ 
9:     for i=1 to nClases do
10:       if registro de clases[i] es optimizable then
11:         precision[i] = Evaluar(GS, ejemplos de TR con clase i)
12:         if precision[i] < menorPrecision then
13:           menorPrecision = precision[i]
14:           claseObj = i
15:         end if
16:       end if
17:     end for
18:     if claseObj = minoritaria y numOptim[claseObj] > 0 then
19:       Mejorar la población GStrial aplicando un algoritmo de Evo-
          lución Diferencial
20:     else
21:       GStrial = GS  $\cup$  { ejemplos aleatorios cogidos de TR pertene-
          cientes a la clase claseObj }
22:       Mejorar la población GStrial aplicando un algoritmo de Evo-
          lución Diferencial
23:     end if
24:     AUCtrial = evaluar población GStrial con TR
25:     if AUCtrial > AUC then
26:       AUC = AUCtrial
27:       GS = GStrial
28:       numero de optimizaciones[i] = 0
29:     else
30:       if claseObj = minoritaria y numOptim[claseObj] < T then
31:          $\triangleright$  T es el número máximo de optimizaciones
32:         registro de clases[claseObj]++
33:       else
34:         registro de clases[claseObj] = no optimizable
35:       end if
36:     end if
37:   end while
38:   return GS
39: end function
```

---

Este algoritmo inicia la población con las instancias más relevantes del conjunto de datos original. Para obtener estas instancias seguimos los siguientes pasos:

- Se aprende un árbol de decisión *C4.5* con el conjunto de datos original.
- Se crean tantos *clústeres* como nodos hoja contenga el árbol. Para cada instancia del conjunto de datos, se determina que hoja del árbol es la que clasifica dicha instancia y se añade a su correspondiente *clúster*.
- Se calculan los centroides de cada *clúster*.
- Para cada *clúster* calculamos la instancia más cercana al centroide. El conjunto de instancias inicial es el formado por la unión de estas instancias.

Con este proceso de selección inicial nos garantizamos coger las instancias más representativas de todo el conjunto de datos. Una vez las tenemos, aplicamos de forma iterativa una versión simplificada del algoritmo *Differential Evolution* [18] para mejorar el conjunto de datos obtenido en la iteración anterior. Este proceso lo hacemos hasta que no podamos mejorar el valor de la función evaluación. En este caso, la función de evaluación es el área bajo la curva *ROC* —métrica que explicaremos en la subsección 7.3.2—.

## 4.10. NearMiss.

El artículo original donde se ha publicado este algoritmo es “*kNN Approach to Unbalanced Data Distribution: A Case Study involving Information Extraction*” escrito por *Jianping Zhang y Inderjeet Mani* [19]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function NM
2:   if version = 1 then
3:     Seleccionamos los elementos mayoritarios cuya distancia media
       a sus tres vecinos de la clase minoritaria más cercanos sea
       menor
4:   if version = 2 then
5:     Seleccionamos los elementos mayoritarios cuya distancia me-
       dia a sus tres vecinos de la clase minoritaria más lejanos sea
       menor
6:   else
7:     Seleccionamos de forma aleatoria una cantidad de elementos
       mayoritarios para cada elemento minoritario
8:   end if
9: end if

```

---

---

```

10:   return Elementos minoritarios de D  $\cup$  Elementos seleccionados
      en los pasos anteriores del algoritmo
11: end function

```

---

La idea de este algoritmo es eliminar los vecinos mayoritarios que sean redundantes. La definición de redundante varía según la versión del algoritmo a ejecutar.

## 4.11. Neighbourhood Cleaning Rule.

El artículo original donde se ha publicado este algoritmo es “*Improving Identification of Difficult Small Classes by Balancing Class Distribution*” escrito por *J. Laurikkala* [20]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function NCL
2:   C = elementos de la clase minoritaria
3:   O = elementos de la clase mayoritaria
4:   Identificar los elementos ruidosos  $A_i$  en O usando ENN
5:   for Clase  $C_i$  en O do
6:     Calculamos los 3 vecinos más cercanos para cada elemento que
      tenga por clase a  $C_i$  y su etiqueta predicha por dichos vecinos
7:     for Conjuntos de 3 vecinos  $v_i$  y etiqueta predicha do
8:       if Si la etiqueta predicha es incorrecta then
9:         Calculamos el número de elementos de las clases asociadas
            a los 3 elementos de  $v_i$ 
10:        Añadimos  $A_2$  los vecinos cuyo número de elementos de su
            clase sea mayor que  $0.5 * \text{número de elementos de la clase}$ 
            minoritaria
11:       end if
12:     end for
13:   end for
14:   return T - ( $A_1 \cup A_2$ )
15: end function

```

---

Este algoritmo primero elimina instancias ruidosas aplicando ENN 4.6. A continuación, para cada instancia, predecimos su clase con los tres vecinos más cercanos. Si la predicción es incorrecta, eliminamos cada uno de sus vecinos si el número de instancias de la clase asociada a cada uno de ellos es mayor que la mitad del número de instancias de la clase minoritaria. De esta manera, lo que hacemos es eliminar vecinos que clasifican de forma errónea otras instancias.

## 4.12. One-Side Selection.

El artículo original donde se ha publicado este algoritmo es “*Addressing the Curse of Imbalanced Training Sets: One-Side Selection*” escrito por *Miroslav Kubat y Stan Matwin* [21]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function OSS(Conjunto de datos S)
2:   C = ejemplos positivos de S y un ejemplo negativo escogido
      aleatoriamente
3:   Clasificar S usando la kNN con k = 1 usando los ejemplos de C.
4:   Añadir a S todos los ejemplos que han sido mal clasificados
5:   Aplicar TomekLink sobre C
6:   Eliminar de C todos los elementos negativos que sean parte de
      un Tomek link
7:   return C
8: end function
```

---

La idea de este algoritmo es detectar que ejemplos no se pueden clasificar usando las instancias minoritarias y una mayoritaria al azar. Una vez se han detectado, aplicamos el algoritmo *Tomek Link* —el cual veremos detallado en la sección 4.14— sobre un conjunto de datos formado por las instancias minoritarias y las instancias mal clasificadas para eliminar instancias ruidosas. El resultado proporcionado por *Tomek Link* es el conjunto de datos reducido.

## 4.13. Random Undersampling.

Este algoritmo es muy simple. Lo único que hacer es eliminar instancias mayoritarias de forma aleatoria hasta que el ratio entre las instancias mayoritarias y las instancias minoritarias sea el especificado en un parámetro, siempre y cuando la distribución de las instancias lo permita.

## 4.14. Tomek Link.

El artículo original donde se ha publicado este algoritmo es “*Two Modifications of CNN*” escrito por *Ivan Tomek* [22]. El pseudocódigo del algoritmo es el siguiente:

---

```

1: function TOMEKLINKS(Conjunto de datos D)
2:   for Cada elemento  $e_i \in D$  do
3:     TomekLinks = {}
4:      $v_i$  = vecino más cercano de  $e_i$  con clase distinta
5:     Calcular la distancia de  $e_i$  a todos los elementos de la clase
        opuesta
```

---

---

```

6:     Calcular la distancia de  $v_i$  a todos los elementos de la clase
opuesta
7:     if El elemento más cercano a  $e_i$  es  $v_i$  then
8:         TomekLinks = TomekLinks  $\cup \{e_i, v_i\}$ 
9:     end if
10:    end for
11:    return D - TomekLinks
12: end function

```

---

Este algoritmo busca eliminar instancias ruidosas. Para ello, lo que hace es calcular el vecino más cercano de la clase opuesta para cada instancia y si ambos son el vecino más cercano entre sí, ambos elementos forman un *tomek-link*. Dado que lo estamos aplicando en técnicas de *undersampling*, solo borramos las instancias pertenecientes a la clase mayoritaria en los *tomek-link*.

## 4.15. Undersampling Based on Clustering.

El artículo original donde se ha publicado este algoritmo es “*Under-Sampling Approaches for Improving Prediction of the Minority Class in an Imbalanced Dataset*” escrito por Show-Jane Yen y Yue-Shi Lee [23]. El funcionamiento principal del algoritmo se basa en la siguiente fórmula:

$$SSize_{MA}^i = (m \times Size_{MI}) \times \frac{Size_{MA}^i / Size_{MI}^i}{\left( \sum_{i=1}^K Size_{MA}^i \right) / Size_{MI}^i} \quad (4.12)$$

Dicha fórmula determina que el número de instancias mayoritarias a seleccionar del *clúster*  $i$  es  $SSize_{MA}^i$ .  $m$  es un ratio —que normalmente vale 1—,  $Size_{MI}$  es el número de instancias minoritarias del conjunto de datos,  $Size_{MA}^i$  es el número de instancias mayoritarias del *clúster*  $i$  y  $Size_{MI}^i$  es el número de instancias minoritarias del *clúster*  $i$ .

El pseudocódigo del algoritmo es el siguiente:

---

```

1: function SBC(Conjunto de datos D)
2:     clusters = kMeans(D)
3:     Calcular el número de ejemplos mayoritarios en cada clúster
     aplicando la ecuación 6.2.18 y seleccionar de forma aleatoria los
     ejemplos mayoritarios de cada clúster
4:     return { Instancias minoritarias }  $\cup$  { Instancias mayoritarias
     seleccionadas en el paso 3.}
5: end function

```

---

La selección de instancias en el paso 3 también se puede hacer aplicando uno de los siguientes criterios:

- *random*: selección puramente aleatoria.
- *NearMiss1*: seleccionamos los elementos mayoritarios cuya distancia media a sus tres vecinos de la clase minoritaria más cercanos sea menor.
- *NearMiss2*: seleccionamos los elementos mayoritarios cuya distancia media a sus tres vecinos de la clase minoritaria más lejanos sea menor.
- *NearMiss3*: seleccionamos de forma aleatoria una cantidad de elementos mayoritarios para cada elemento minoritario.
- *MostDistant*: seleccionamos los ejemplos cuya distancia media a los  $N$  vecinos del *clúster*  $i$  de la clase minoritaria más cercanos sea mayor.
- *MostFar*: seleccionamos los ejemplos cuya distancia media a todos los vecinos de la clase minoritaria sea mayor.

La idea de este algoritmo es crear un conjunto de datos balanceado en cada *clúster* y devolver la unión de todos los subconjuntos balanceados formado en cada *clúster*. Haciendo esto, nos quedamos con los elementos más representativos de ambas clases y en una cantidad similar, garantizando así que obtenemos un conjunto balanceado al final del proceso.



# Capítulo 5

## Introducción a Scala.

*Scala* [24] es un lenguaje de programación multi-paradigma. Su nombre viene de la concatenación de dos palabras: *Scalable* y *Language*. Es un lenguaje de programación orientado a objetos —ya que todo valor es un objeto— pero también integra características de los lenguajes funcionales. Su implementación actual corre sobre la máquina virtual de *Java* y, por lo tanto, es totalmente compatible con dicho lenguaje.

Junto con *Scala* se ha usado *sbt* [25] —acrónimo de *Simple Build Tool*—. Como su nombre indica, es una herramienta que permite compilar y ejecutar proyectos de *Scala* —y de *Java*—. También permite definir un fichero *build.sbt* en el que se pueden incluir dependencias y la metainformación del proyecto —como sucede con otras herramientas de desarrollo, por ejemplo *Maven* [26] y su fichero *pom.xml*—. El uso de *sbt* facilita las tareas de compilación, gestión de dependencias y ejecución de este proyecto. También se ha usado para generar la documentación de este proyecto, tal y como se explica en la sección 6.2.

### 5.1. ¿Por qué Scala?

#### 5.1.1. Patrón de diseño mixto.

El potencial de *Scala* reside en la combinación de los dos paradigmas de programación, orientado a objetos y el funcional. El paradigma funcional está cogiendo mucha fuerza en los últimos años con el tema del *Big Data*. El *Big Data* requiere de código eficiente y limpio, y esto se puede conseguir con la programación funcional. *Scala* nos permite conseguir nuestro objetivo gracias al uso de valores inmutables, colecciones inmutables, funciones sin efectos colaterales y funciones de primera clase, es decir, las funciones son tratadas como cualquier otra variable.

### 5.1.2. Simplicidad y elegancia.

Otro potencial de *Scala* reside en la elegancia y simplicidad para escribir código. Al ser un lenguaje con características funcionales, se pueden hacer muchas operaciones en muy pocas líneas de código. Por ejemplo, crear una matriz de tamaño diez por diez con valores aleatorios entre cero y uno es tan sencillo como:

```
1 val r = new scala.util.Random
2 val data = (0 until 10).map(_ => (0 until 10).map(_ => r.nextDouble))
```

Por ejemplo, definamos una clase *Persona* con la funcionalidad básica para poder almacenar el nombre y el apellido de una persona. El código en *Java* sería el siguiente:

```
1 public class Persona {
2     private final String nombre;
3     private final String apellido;
4     public Person(String nombre, String apellido) {
5         this.nombre = nombre;
6         this.apellido = apellido;
7     }
8     public String getNombre() {
9         return nombre;
10    }
11    public String getApellido() {
12        return apellido;
13    }
14 }
```

El equivalente a esta clase en *Scala* es el siguiente:

```
1 class Persona(val nombre: String, val apellido: String)
```

Como podemos ver, con mucho menos código podemos hacer lo mismo. Además, si hubiésemos definido la clase en *Scala* como:

```
1 case class Persona(val nombre: String, val apellido: String)
```

tendríamos implementadas las funcionalidades de copia y comparación, tal y como podemos ver en la documentación de las clases *case* [27].

### 5.1.3. Escalabilidad.

*Scala* es, tal y como indica su nombre, muy escalable, es simple para escribir pequeños *scripts* personales pero muy potente para escribir grandes proyectos, como puede ser el proyecto explicado en este documento.

#### 5.1.4. Modificadores de visibilidad.

En *Scala* tenemos un mayor control de la visibilidad que en *Java*. Tenemos los siguientes modificadores:

- *Publico*: Es la visibilidad por defecto en *Scala*. El objeto con dicha visibilidad es accesible desde cualquier otro objeto.
- *Protegido*: Identificado por la palabra reservada *protected*. Visibles por los tipos que lo definen, tipos que heredan y tipos anidados dentro del paquete o subpaquetes donde se define.
- *Privado*: Identificado por la palabra reservada *private*. Visibles por los tipos que lo definen y tipos anidados dentro del paquete donde se define.
- *Protegido[scope]*: Identificado por la palabra reservada *protected [scope]*. Visibilidad limitada por *scope*, el cual puede ser un paquete, una clase o el propio *this*, es decir, la propia clase donde se define.
- *Privado[scope]*: Identificado por la palabra reservada *private [scope]*. Visibilidad limitada por *scope*, el cual puede ser un paquete, una clase o el propio *this*, es decir, la propia clase donde se define. Las clases que heredan de esta clase no pueden acceder a dichos objetos.

#### 5.1.5. Paralelismo.

*Scala* tiene implementadas colecciones paralelas. Estas colecciones permiten ejecutar, de forma transparente para el usuario, código en paralelo sin tener que invertir esfuerzo en programar el paralelismo. Por ejemplo, podemos imprimir el contenido de un objeto *Array* de la siguiente forma:

```
1 Array(0, 1, 2, 3, 4, 5).foreach(println)
```

obteniendo así el resultado esperado: 0, 1, 2, 3, 4 y 5. Si quisieramos hacerlo de forma paralela, solo debemos convertir el objeto *Array* en un array paralelo, es decir, un *ParArray*:

```
1 Array(0, 1, 2, 3, 4, 5).par.foreach(println)
```

obteniendo así como resultado: 0, 3, 5, 4, 2 y 1.

#### 5.1.6. Ausencia de Scala.

Finalmente, otro motivo por el cual se ha elegido *Scala* como lenguaje para este proyecto es que no hay ninguna biblioteca tan compleja y que abarque tantos algoritmos de clasificación no balanceada escrita en *Scala*.



# Capítulo 6

## Descripción técnica.

En este capítulo vamos a ver un par de aspectos técnicos empleados en el desarrollo del proyecto y como un pequeño manual de usuario que permita recrear parte de los experimentos mostrados en este documento.

### 6.1. Indices.

Como se comentó en el capítulo 3, uno de los requisitos de este proyecto es mantener los datos intactos siempre que se pueda y, para ello, los algoritmos trabajan con índices en vez de con los propios datos. *Scala* proporciona una manera muy elegante de acceder a los elementos de una colección dado un índice. Por ejemplo, para acceder a los elementos *0*, *3* y *4* de un vector con los elementos *0*, *10*, *20*, *30*, *40*, *50*, solo debemos usar la función *map*:

```
1 scala> val indice = Array(0, 3, 4)
2 scala> val data = Array(0, 10, 20, 30, 40, 50)
3 scala> indice map data
4 res0: Array[Int] = Array(0, 30, 40)
```

Otra función muy interesante para recorrer una colección es *zipWithIndex*. Dicha función aplicada sobre una colección devuelve una colección donde los elementos están formados por el elemento original y su índice dentro de dicha colección. Veamos un ejemplo:

```
1 scala> val data = Array(0, 10, 20, 30)
2 scala> data.zipWithIndex
3 res0: Array[(Int, Int)] = Array((0,0), (10,1), (20,2), (30,3))
```

Jugando con estas dos funciones, el trabajo con índices es bastante sencillo y elegante, siendo esta la metodología usada en este proyecto.

## 6.2. Implementación de los algoritmos.

En esta sección vamos a comentar los aspectos de implementación de los algoritmos desarrollados en esta biblioteca. Cabe destacar que todos los algoritmos se ejecutan llamando al método *sample* de la clase correspondiente. Este método varía en los argumentos que recibe —ya que cada uno de ellos necesita unos argumentos concretos—. Para facilitar el uso de esta biblioteca al usuario todos los argumentos usados en todos los algoritmos tienen un valor por defecto, para no forzar a que el usuario deba entender o conocer el algoritmo para así poder decidir que valores usar. Todos los algoritmos devuelven el mismo objeto *Data* que se ha usado como entrada pero con el resultado calculado asignado a los atributos *\_resultData*, *\_resultClasses* y *\_index*.

Todos los algoritmos nos permiten obtener información acerca de la ejecución en un fichero de texto gracias a la clase *Logger* implementada en el paquete *io*. Para ellos, los algoritmos esperan un parámetro llamado *file* de tipo *Option[String]*. Un *Option* en *Scala* puede tomar dos valores: *None* para indicar la ausencia de valor o *Some(X)* donde *X* es un objeto del tipo definido por el *Option*. Si a los algoritmos le pasamos *None* en dicho parámetro no se realizará ninguna operación de *log*. Si pasamos un *Option(nombreFichero)* se realizará almacenará información acerca de la ejecución del algoritmo en un fichero —que se creará si no existe— llamado *nombreFichero*.

En esta sección vamos a ver las cabeceras y los distintos argumentos de cada algoritmo. Para facilitar esta documentación al usuario, se ha desarrollado una página web con la documentación del proyecto. Se ha utilizado *Scaladoc* [28] para documentar el código. Una vez está todo el código documentado, solo debemos ejecutar *sbt doc* para generar una página web con toda la documentación. Dicha página web está alojada en *GitHub Pages* [29] y se puede acceder a ella en la dirección <https://nestorrv.github.io>.

### 6.2.1. Logger.

*Logger* es una clase diseñada para almacenar información de como está funcionando la ejecución de un algoritmo y, al finalizar el mismo, guardarla en un fichero de texto. En dicho fichero se almacena información acerca del tamaño del conjunto de datos, de su *Imbalanced Ratio*, del tiempo de ejecución y cualquier información relevante acerca del algoritmo.

### 6.2.2. Data.

Como se comentó en la sección 3.1 todos los algoritmos hacen uso de la clase *Data* para poder almacenar la información necesaria para el funcio-

namiento de los algoritmos. Dicha clase es muy sencilla, solo contiene los atributos necesarios para almacenar toda la información y varios métodos para acceder a los atributos. La información almacenada por *Data* es:

- *\_nominal*: objeto del tipo *Array[Int]* indicando que atributos —columnas de la matrix de datos— son atributos nominales.
- *\_originalData*: objeto del tipo *Array[Array[Any]]* que contiene el conjunto de datos original.
- *\_originalClasses*: objeto del tipo *Array[Array[Any]]* que contiene las clases asociadas a *\_originalData*.
- *\_fileInfo*: objeto del tipo *FileInfo*. *FileInfo* es una clase implementada en mi biblioteca que contiene información auxiliar del fichero leído, como puede ser el separador de elementos, la cadena que indica elemento vacío, la cabecera del fichero —incluyendo la información de los atributos representados por *ARFF* [6] si es que existe—, *etcétera*.
- *\_processedData*: objeto del tipo *Array[Array[Double]]* que representa los datos procesados para poder ser usados por mis algoritmos.
- *\_resultData*: objeto del tipo *Array[Array[Any]]* en el cual se escribe el conjunto resultante de la ejecución de mis algoritmos.
- *\_resultClasses*: objeto del tipo *Array[Any]* en el cual se escriben las clases asociadas a *\_resultData*.
- *\_index*: objeto del tipo *Array[Int]* que representa los índices de los elementos originales que han sido preservados por mis algoritmos.

### 6.2.3. Algorithm.

En la sección 3.1 también vimos que cada algoritmo ha sido implementado en una clase distinta, pero todas heredan de *Algorithm*. *Algorithm* es un *trait*. Un *trait* en *Scala* [30] es el equivalente a una interfaz en *Java*. Permite definir un tipo de objeto especificando el comportamiento mediante métodos. Los *traits* no pueden tener argumentos en el constructor pero, a diferencia de las interfaces de *Java*, si pueden tener métodos implementados.

La información almacenada por *Algorithm* es:

- *data*: objeto de la clase *Data* comentada en la sección 6.2.2.
- *seed*: objeto del tipo *Long* que representa la semilla a usar para inicializar el generador de números aleatorios. Si no se proporciona ninguna, se usan los milisegundos del sistema como semilla.

- *minorityClass*: objeto del tipo *Any* que representa cual es la clase minoritaria para el usuario. Si no se proporciona ninguna, se asigna a -1 para funcionamiento interno de los algoritmos.
- *x*: objeto del tipo *Array[Array[Double]]* que contiene los datos sin las clases.
- *y*: objeto del tipo *Array[Any]* que contiene las clases del conjunto de datos.
- *logger*: objeto del tipo *Logger* implementado en mi biblioteca. Es una clase encargada de ir almacenando los mensajes de *log* especificados por los algoritmos y escribirlos en un fichero de datos si el usuario ha activado esa opción.
- *counter*: objeto del tipo *Map[Any, Int]* el cual almacena el número de instancias que hay en cada clase.
- *untouchableClass*: objeto del tipo *Any* usado para representar la clase minoritaria del conjunto de datos. Si el objeto *minorityClass* está especificado por el usuario, *untouchableClass* toma su valor. En caso contrario, se calcula como la clase con menor número de instancias.
- *index*: objeto del tipo *List[Int]* que representa un índice barajado de los datos, para aleatorizar el orden de los mismos.

#### 6.2.4. Clase BalanceCascade

Este algoritmo no se ha implementado siguiendo fielmente las indicaciones del *paper* original sino que se ha implementado la versión propuesta en la biblioteca *imbalanced-learn* [3]. El pseudocódigo implementado es el siguiente:

---

```

1: function BALANCE CASCADE(conjunto de datos D, ratio)
2:   search = true
3:   mask = {true, ..., true} // size(mask) == size(conjunto de datos)
4:   while search do
5:     contadorClases = numero de instancias de cada clase de los elementos que tienen el valor de mask a true
6:     for ci ∈ contadorClases do
7:       mismaClase = instancias de la misma clase que ci
8:       if ci no es la clase minoritaria then
9:         targetIndex = instancias de mismaClase y que su valor de mask está a true
10:        instanciasMaj = instanciasMaj ∪ targetIndex

```

---

---

```

11:      else
12:          instanciasMinoritarias = mismaClase // guardamos todas
13:          las instancias minoritarias
14:      end if
15:  end for
16:  numeroSubSetsCreados += 1
17:  if numeroSubSetsCreados == maxSubSets then
18:      search = false
19:  end if
20:  subset = targetIndex ∪ instanciasMinoritarias
21:  prediction = predecimos las clases haciendo k-fold
22:  clasificadas = instancias bien clasificadas
23:  ponemos los valores de mask de las instancias bien clasificadas
24:  a false
25:  contadorClasesNuevo = numero de instancias de cada clase de
26:  los elementos que tienen el valor de mask a true
27:  search = false si alguna clase de contadorClasesNuevo tiene me-
28:  nor número de instancias que la clase minoritaria
29: end while
30: histogramaMayoritarias = histograma del número de ocurrencias
31: de cada instancia mayoritaria en instanciasMaj
32: N = |instancias de la clase minoritaria| * ratio
33: salida = instancias de la clase minoritaria ∪ N instancias más
34: ocurrientes basándonos en histogramaMayoritarias
35: return salida
36: end function

```

---

Este algoritmo va creando los mismos subconjuntos que se crean en el conjunto original hasta que se hayan creado todos los posibles. Estos conjuntos se crean usando un clasificador —usando la técnica del vecino más cercano— así que mantiene la idea de usar un clasificador del *paper* original. Finalmente nos quedamos con los elementos que más veces hayan aparecido en los *subsets* y las instancias minoritarias. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample( file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN, k: Int = 3, nMaxSubsets: Int = 5, nFolds: Int
  = 5, ratio: Double = 1.0): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.

- *k*: número de vecinos a usar al realizar la predicción de la línea 20 del pseudocódigo mostrado en esta sección.
- *nMaxSubsets*: número máximo de subsets a crear.
- *nFolds*: número de folds a crear al realizar la predicción de la línea 20 del pseudocódigo mostrado en esta sección.
- *ratio*: ratio indicando el número de instancias mayoritarias a coger. Se cogerán  $|numero\ instancias\ minoritarias| * ratio$ .

### 6.2.5. Clase ClassPurityMaximization

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.

### 6.2.6. Clase ClusterOSS

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN, k: Int = 3, numClusters: Int = 15, restarts:
  Int = 5, minDispersion: Double = 0.0001, maxIterations: Int = 100)
  : Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.
- *k*: número de vecinos a usar al realizar la predicción de la línea 20 del pseudocódigo mostrado en esta sección.
- *numClusters*: número de *clústeres* a crear.

- *restarts*: número de veces a aplicar el algoritmo de clustering —el algoritmo se queda el mejor resultado de todas las ejecuciones—.
- *minDispersion*: si la dispersión tras una iteración del algoritmo *kMeans* [12] es menor que este valor, se detiene la ejecución.
- *maxIterations*: número máximo de iteraciones permitidas para el algoritmo *kMeans*.

#### 6.2.7. Clase CondensedNearestNeighbor

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.

#### 6.2.8. Clase EasyEnsemble

Este algoritmo no se ha implementado siguiendo fielmente las indicaciones del *paper* original sino que se ha implementado la versión propuesta en la biblioteca *imbalanced-learn*. El pseudocódigo implementado es el siguiente:

---

```
1: function EASYENSEMBLE(conjunto de datos D, nVeces, ratio)
2:   mayoritarias = {}
3:   for i ← nVeces do
4:     aux = coger aleatoriamente |instancias minoritarias| * ratio
      instancias mayoritarias
5:     mayoritarias = mayoritarias ∪ aux
6:   end for
7:   histogramaMayoritarias = histograma del número de ocurrencias
      de cada instancia mayoritaria en mayoritarias
8:   N = |instancias de la clase minoritaria| * ratio
9:   salida = instancias de la clase minoritaria ∪ N instancias más
      ocurrientes basándonos en histogramaMayoritarias
10:  return salida
11: end function
```

---

La idea es hacer una mezcla se subconjuntos pero generados de forma aleatoria. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, ratio: Double = 1.0, replacement:
    Boolean = false, nTimes: Int = 5): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *ratio*: ratio indicando el número de instancias mayoritarias a coger. Se cogerán  $|numero\ instancias\ minoritarias| * ratio$ .
- *replacement*: indica si aplicar reemplazo o no a la hora de seleccionar los elementos mayoritarios en cada subconjunto. Si se aplica, una instancia puede ser seleccionada múltiples veces para el mismo subconjunto.
- *nTimes*: indica el número de subconjuntos a crear.

### 6.2.9. Clase EditedNearestNeighbor

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
    Distances.EUCLIDEAN, k: Int = 3): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.
- *k*: número de vecinos a usar al realizar la predicción de las etiquetas.

### 6.2.10. Clase EvolutionaryUnderSampling

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, populationSize: Int = 50,
    maxEvaluations: Int = 1000, algorithm: String = "EBUSMSGM",
    distance: Distances.Distance = Distances.EUCLIDEAN, probHUX:
    Double = 0.25, recombination: Double = 0.35, prob0tol: Double =
    0.05): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *populationSize*: número de individuos que contiene la población.
- *maxEvaluations*: número máximo de evaluaciones a realizar por el algoritmo.
- *algorithm*: cadena de texto indicando la versión del algoritmo a ejecutar. Las distintas versiones están explicadas en la sección 4.7.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.
- *probHUX*: probabilidad de cambiar un gen de cero a uno, utilizado en el proceso de cruce.
- *recombination*: umbral de recombinación, usado en el proceso de reinicialización.
- *prob0to1*: probabilidad de cambiar un gen de cero a uno, utilizado en el proceso de reinicialización.

### 6.2.11. Clase InstanceHardnessThreshold

Este algoritmo no se ha implementado siguiendo fielmente las indicaciones del *paper* original sino que se ha implementado la versión propuesta en la biblioteca *imbalanced-learn*. El pseudocódigo implementado es el siguiente:

---

```

1: function INSTANCEHARDNESSTHRESHOLD(conjunto de datos D, N)
2:   particionamiento = aplicar k-fold y crear las N particiones
3:   probabilidades = {0.0, ..., 0.0} // size(probabilidades) == size(D)
4:   for conjunto de train y test ∈ particionamiento do
5:     c4.5 = entrenar un árbol de decisión C4.5 con el conjunto de train
6:     prob = probabilidad de cada instancia de test de pertenecer a la
      clase de dicha instancia
7:     guardar en probabilidades la probabilidad de cada instancia
8:   end for
9:   indiceFinal = {}
10:  for  $c_i \in$  clasesDisponibles do
11:    if  $c_i \neq$  claseMinoritaria then
12:      nSamples = numero de instancias de la clase  $c_i$ 
13:      targetIndex = indice de las instancias que tienen la clase  $c_i$ 
14:      targetProbs = probabilidades de las instancias seleccionadas
          por targetIndex

```

---

---

```

15:     n = número de instancias que tienen la clase  $c_i$ 
16:     percentil = (1.0 - (nSamples / n)) * 100.0
17:     corte = (targetProbs.length - 1) * (percentil / 100.0)
18:     umbral = ordenar targetProbs y coger el elemento en la
19:     posición corte
20:     indexTargetClass = instancias que tienen la clase  $c_i$  cuya
21:     probabilidad de pertenecer  $c_i$  supera el umbral
22:   else
23:     indexTargetClass = indice de las clases minoritarias
24:   end if
25:   indiceFinal = indiceFinal  $\cup$  instancias seleccionadas por
      indexTargetClass
26: end for
27: end function

```

---

La idea detrás de esta implementación es calcular la probabilidad de cada instancia de pertenecer a cada clase y quedarnos con aquellos elementos cuya probabilidad supera un umbral. Dicho umbral viene calculado por el percentil calculado sobre las instancias objetivo. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, nFolds: Int = 5): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *nFolds*: número de *folds* a crear al aplicar *cross-validation*.

### 6.2.12. Clase IterativeInstanceAdjustmentImbalancedDomains

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, iterations: Int = 100, strategy:
      Int = 1, randomChoice: Boolean = true): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *iterations*: número máximo de iteraciones a realizar por el algoritmo.
- *strategy*: indica que estrategia a seguir en el proceso de mutación del algoritmo *Differential Evolution*.

- *randomChoice*: indica si se selecciona un nuevo individuo de forma aleatoria o no en la ejecución del algoritmo.

### 6.2.13. Clase NearMiss

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN, version: Int = 1, nNeighbours: Int = 3, ratio
  : Double = 1.0): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.
- *version*: indica que versión del algoritmo ejecutar. Las distintas versiones las podemos ver en la sección 4.10.
- *nNeighbours*: numero de vecinos de la clase mayoritaria a coger por cada instancia minoritaria. Solo se usa si la versión usada es la 3.
- *ratio*: ratio indicando el número de instancias mayoritarias a coger. Se cogerán  $|numero\ instancias\ minoritarias| * ratio$ .

### 6.2.14. Clase NeighbourhoodCleaningRule

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN, k: Int = 3, threshold: Double = 0.5): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.
- *k*: número de vecinos a usar al aplicar la regla *k-NN*.
- *threshold*: una clase será reducida si el número de instancias asociadas a dicha clase es mayor que  $|numero\ instancias\ minoritarias| * threshold$ .

### 6.2.15. Clase OneSideSelection

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.

### 6.2.16. Clase RandomUndersampling

Este algoritmo se ha implementado de forma intuitiva, ya que es el más sencillo de toda la biblioteca y no sigue ningún *paper*. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, ratio: Double = 1.0, replacement:
  Boolean = false): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *ratio*: ratio indicando el número de instancias mayoritarias a coger. Se cogerán  $|numero\ instancias\ minoritarias| * ratio$ .
- *replacement*: indica si aplicar reemplazo o no a la hora de seleccionar los elementos mayoritarios. Si se aplica, una instancia puede ser seleccionada múltiples veces.

### 6.2.17. Clase TomekLink

Este algoritmo se ha implementado como indica el *paper* original. Se ha añadido un argumento extra que permite decidir que instancias eliminar una vez se han creado los *tomek links*. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, distance: Distances.Distance =
  Distances.EUCLIDEAN, ratio: String = "not minority"): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *distance*: indica el tipo de distancia a usar. Es un valor del enumerado *Distances* implementado en el paquete *util* de esta biblioteca.
- *ratio*: indica que instancias se deben borrar de los *tomek links*. Si su valor es *all*, se borrarán todas las instancias; si su valor es *minority*, se borrarán solo las instancias de la clase minoritaria y si su valor es *not minority*, se borrarán las instancias de las clases que no sean la minoritaria.

### 6.2.18. Clase UndersamplingBasedClustering

Este algoritmo se ha implementado como indica el *paper* original. La cabecera que ejecuta este algoritmo es la siguiente:

```
1 sample(file: Option[String] = None, method: String = "random", m:
  Double = 1.0, k: Int = 3, numClusters: Int = 50, restarts: Int =
  1, minDispersion: Double = 0.0001, maxIterations: Int = 200): Data
```

Los argumentos son:

- *file*: si está definido, se guardará un fichero de *log* —usando la clase *Logger*— en el fichero indicado por el valor de dicho parámetro.
- *method*: indica que método a ejecutar a la hora de seleccionar las instancias mayoritarias. Los posibles valores son *random*, *NearMiss1*, *NearMiss2*, *NearMiss3*, *MostDistant* y *MostFar*. El significado de cada uno de ellos está explicado en la sección 4.10.
- *m*: ratio usado en el cálculo de  $SSize_{MA}^i$  explicado en la equación .
- *k*: número de vecinos a usar al aplicar la regla *k-NN*.
- *numClusters*: número de *clústeres* a crear.
- *restarts*: número de veces a aplicar el algoritmo de clustering —el algoritmo se queda el mejor resultado de todas las ejecuciones—.
- *minDispersion*: si la dispersión tras una iteración del algoritmo *kMeans* es menor que este valor, se detiene la ejecución.
- *maxIterations*: número máximo de iteraciones permitidas para el algoritmo *kMeans*.

### 6.3. Funciones de entrada y salida.

Esta biblioteca tiene una clase destinada a la lectura de datos y otra a la escritura de datos, facilitando así la ejecución de los algoritmos y el guardado del resultado de los mismos. Estás clases son *Reader* y *Writer*, implementadas dentro del paquete *io*. La clase *Reader* lee los datos de un fichero de datos y crear un objeto de tipo *Data* listo para ser pasado directamente a un algoritmo. Una vez se ha aplicado el algoritmo, se puede pasar el objeto *Data* a la clase *Writer* para escribirlo en un fichero de salida.

### 6.4. Manual de usuario.

En esta sección vamos a comentar los pasos que tendría que hacer un usuario para hacer uso de esta biblioteca. Lo primero que debe hacer el usuario es leer un conjunto de datos para poder ser usado. Supongamos que tiene un conjunto de datos en formato *ARFF* y otro en formato *csv* [31]. Para leer dichos conjuntos debemos hacer lo siguiente:

```

1 import undersampling.io.Reader
2
3 val reader = new Reader
4
5 val csvData: Data = reader.readDelimitedText(file = pathToFile)
6 val arffData: Data = reader.readArff(file = pathToFile)

```

Una vez tenemos los conjuntos de datos, solo debemos decidir que algoritmo queremos aplicar y llamar a su método *sample*. Veamos como aplicar, por ejemplo, el algoritmo *NCL* [20] —para todos los algoritmos el proceso es el mismo—. Lo vamos sobre ambos conjuntos de datos, aunque el proceso es el mismo. Supongamos que queremos crear un fichero de *log* para obtener la información de ejecución del algoritmo y que queremos que este fichero de *log* se llame *milogX*, donde *X* es el formato del fichero de datos de entrada:

```

1 import undersampling.core.NeighbourhoodCleaningRule
2
3 val nclCSV = new NeighbourhoodCleaningRule(csvData, seed = 0L)
4 val resultCSV: Data = nclCSV.sample(file = Option("milogCSV.log"))
5
6 val nclARFF = new NeighbourhoodCleaningRule(arffData, seed = 0L)
7 val resultARFF: Data = nclARFF.sample(file = Option("milogARFF.log"))

```

En este ejemplo hemos usado los valores por defecto de todos los parámetros del algoritmo para facilitar el uso del mismo, pero en la página de la documentación —<https://nestorrv.github.io>— podemos ver que argumentos espera cada algoritmo así como una descripción de que es cada uno de ellos.

Finalmente, podemos volcar el resultado del algoritmo a un fichero de

datos llamado  $resultX.X$ , donde  $X$  es el formato del fichero de datos de entrada. Para ello, solo debemos ejecutar:

```
1 import undersampling.io.Writer  
2  
3 val writer: Writer = new Writer  
4  
5 writer.writeDelimitedText(file = "salidaCSV.csv", data = resultCSV)  
6 writer.writeArff(file = "salidaARFF.arff", data = resultARFF)
```

Con estos sencillos pasos tenemos un conjunto de datos reducido por el algoritmo *NCL* almacenado en un fichero de datos con el mismo formato que el conjunto de datos original y un fichero de *log* donde ver información sobre como ha ido el proceso.



# Capítulo 7

## Experimentación.

### 7.1. Introducción.

Todos los algoritmos implementados en esta biblioteca son algoritmos de preprocessamiento, es decir, algoritmos que se aplican sobre los conjuntos de datos previo a los algoritmos de aprendizaje automático. Los experimentos realizados implementan los siguientes pasos:

- Aplicar los distintos algoritmos de *undersampling* a los conjuntos de datos.
- Aprender un clasificador *C4.5* [4] y un clasificador *SVM* [5] para los conjuntos de datos originales, aplicando la técnica *k-fold cross validation* con  $k=5$ .
- Aprender un clasificador *C4.5* y un clasificador *SVM* para los conjuntos de datos reducidos por los algoritmos, aplicando la técnica *k-fold cross validation* con  $k=5$ . Para ello, dividimos el conjunto de datos original aplicando la técnica *k-fold cross validation* descrita en la sección 7.3. A continuación, aplicamos el algoritmo de *undersampling* correspondiente al conjunto de entrenamiento —*train*— y probamos a clasificar el conjunto de prueba —*test*—, el cual no ha sido reducido por el algoritmo de *undersampling*.
- Comparar las distintas métricas obtenidos de los clasificadores para los conjuntos de datos originales y para los conjuntos de datos reducidos.

### 7.2. undersampling

Lo primero a realizar es aplicar los quince algoritmos sobre los conjuntos de datos seleccionados. Estos conjuntos de datos han sido obtenidos de la página web de *Keel* [32] [33]. Los conjuntos seleccionados son conjuntos

clásicos empleados en problemas de clasificación binaria. Algunos de ellos tiene una gran diferencia entre el número de instancias positivas —instancias pertenecientes a la clase de interés— y el número de instancias negativas —instancias asociadas a una clase distinta de la clase de interés—. Los conjuntos de datos seleccionados han sido: *ecoli-0-vs\_1*, *ecoli1*, *ecoli2*, *ecoli3*, *glass-0-1-2-3-vs\_4-5-6*, *glass0*, *glass1*, *glass6*, *haberman*, *iris0*, *new-thyroid1*, *new-thyroid2*, *page-blocks0*, *pima*, *segment0*, *vehicle0*, *vehicle1*, *vehicle2*, *vehicle3*, *wisconsin*, *yeast1*, *yeast3*. Se ha modificado el formato de los conjuntos de datos para convertirlos en formato *ARFF* [6] de *WEKA* [7] [8], formato el cuál es capaz de leer esta biblioteca —entre otros—.

En problemas de clasificación no balanceada se define una métrica denominada *Imbalanced Ratio* —o *ratio de desbalanceo*—. Dicha medida se calcula como:

$$IR = \frac{|\text{instancias negativas}|}{|\text{instancias positivas}|} \quad (7.1)$$

Cuanto más desbalanceado está el conjunto, mayor es dicho valor.

### 7.3. Clasificación.

La técnica *k-fold cross validation* es un proceso muy típico usado en *Machine Learning*. Tal y como se comentó en la introducción, nuestro objetivo es aprender las características de un conjunto de datos para poder predecir resultados futuros. Para simular este proceso y ver el rendimiento de los algoritmos lo que hacemos es particionar el conjunto de datos que tenemos disponible en un conjunto con el que entrenamos nuestro algoritmo —al que denominaremos conjunto de entrenamiento o conjunto de *train*— y un conjunto con el que probar su rendimiento —al que denominaremos conjunto de *test*—. Si realizamos este particionamiento una única vez, puede darse el caso en el que hayamos dado con la mejor partición y los resultados obtenidos no sean fiables. Para evitar esto, se aplica la técnica *k-fold cross validation*. La idea es dividir en *k* el conjunto de datos original e ir cambiando que partición se usa como conjunto de *test* y usar el resto como conjunto de *train*. En nuestro caso hemos usado *k=5* —usando así un ochenta por ciento del conjunto de datos original como conjunto de *train* y un veinte por ciento como conjunto de *test*—. Una representación gráfica del proceso es la siguiente:

Iteracion 1	Test	Train	Train	Train	Train
Iteracion 2	Train	Test	Train	Train	Train
Iteracion 3	Train	Train	Test	Train	Train
Iteracion 4	Train	Train	Train	Test	Train
Iteracion 5	Train	Train	Train	Train	Test

Para poder comparar la calidad de los conjuntos de datos generados por los algoritmos de esta biblioteca, se ha entrenado un clasificador *C4.5* [4] y una *SVM* [5]. Ambos clasificadores son los implementados en la librería *WEKA* [7] [8]. Dichos clasificadores han sido entrenados para cada una de las cinco particiones de *train* comentadas anteriormente y probados contra la correspondiente partición de *test*. Este proceso se ha hecho para los conjuntos de datos originales y para los conjuntos de datos reducidos por cada algoritmo. Para poder comparar los resultados se han usado las 3 métricas descritas a continuación.

### 7.3.1. Porcentaje de acierto.

Es la métrica más usada en este tipo de problemas y, probablemente, la más fácil de calcular y obtener. Este tipo de algoritmos pretende predecir un comportamiento y que mejor manera de medir como de bueno es el algoritmo que calculando como de buenos es prediciendo nuevas instancias. Para obtener esta métrica solo debemos predecir las clases del conjunto de *test*, compararlas con las etiquetas reales de dicho conjunto y calcular que porcentaje son correctas. A mayor porcentaje, mejor es el algoritmo, ya que consigue predecir un mayor número de instancias.

### 7.3.2. AUC.

Acrónimo de *Area Under Curve*, o *Area Bajo la Curva*. En este proyecto se ha usado la curva *ROC* —acrónimo de *Receiver Operating Characteristic*, *Característica Operativa del Receptor* es español— [34]. La curva ROC se obtiene dibujando el ratio de verdaderos positivos —*True Positive Rate*, *TPR*— frente al ratio de falsos positivos —*False Positive Rate*, *FPR*— para

diferentes valores de los parámetros empleados en el algoritmo. El ratio de verdaderos positivos se puede calcular aplicando la siguiente fórmula:

$$TPR = TP/(TP + FN) \quad (7.2)$$

donde  $TP$  representa los *True Positives* —instancias positivas que se han clasificado correctamente como positivas— y  $FN$  representa los *False Negatives* —instancias positivas que se han clasificado incorrectamente como negativas—. El ratio de falsos positivos se puede calcular aplicando la siguiente fórmula:

$$FPR = FN/(FP + TN) \quad (7.3)$$

donde  $FN$  representa los *False Negatives* —instancias positivas que se han clasificado incorrectamente como negativas—,  $FP$  representa los *False Positives* —instancias negativas que se han clasificado incorrectamente como positivas— y  $TN$  representa los *True Negatives* —instancias negativas que se han clasificado correctamente como negativas—.

La gráfica de la curva ROC tiene la siguiente forma <sup>1</sup>:

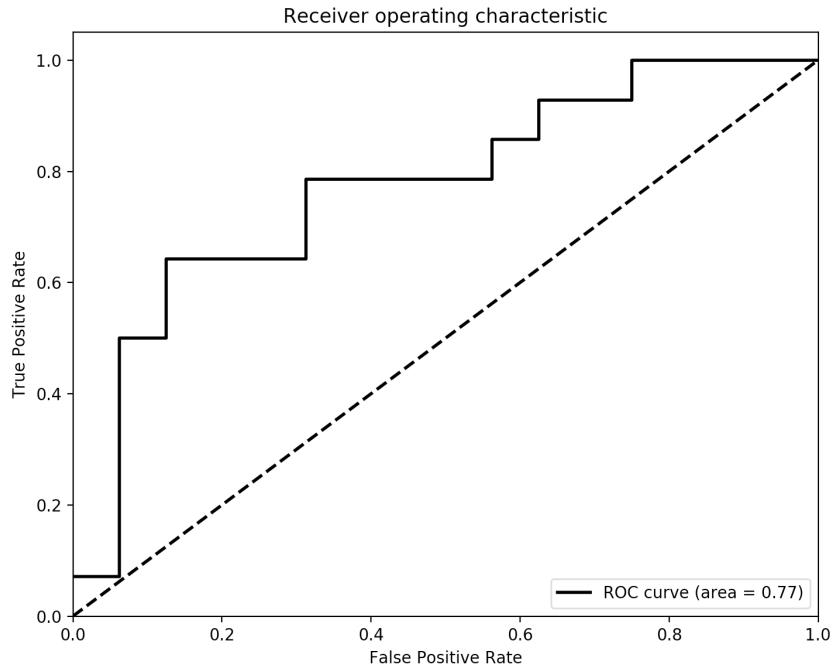


Figura 7.1: Ejemplo curva ROC.

Para ver si un clasificador es bueno podemos calcular el área bajo la curva ROC. A mayor valor de área, mejor es el clasificador. En la imagen

---

<sup>1</sup>Imagen creada por mí usando *Python* [35] y *scikit-learn*. [36]

7.1 podemos ver una línea discontinua. Dicha línea representa un área de 0.5. En dicha imagen también podemos ver que el área calculada es de 0.77, un valor bastante aceptable.

### 7.3.3. Media geométrica.

La media geométrica [37] se define como:

$$g = \sqrt{a^+ \cdot a^-} \quad (7.4)$$

donde  $a^+$  denota la precisión en las instancias positivas —esto es  $TPR$ — y  $a^-$  denota la precisión en las instancias negativas —esto es  $TNR$ —.  $TPR$  calcula como hemos visto en la ecuación 7.2 y  $TNR$  se calcula de la siguiente forma:

$$TNR = \frac{TP}{TP + FN} \quad (7.5)$$

Esta métrica une dos objetivos; busca maximizar la precisión a la vez que equilibrar el tamaño de ambas clases.

## 7.4. Resultados numéricos.

A continuación se muestran varias tablas donde iremos analizando los resultados obtenidos por los distintos algoritmos. Como se ha comentado en la sección 7.3 se ha usado *k-fold* y la cantidad de datos recogida es abrumadora. Es por ello que, a continuación, se muestran las tablas con los valores medios de las cinco ejecuciones. Dichas tablas tienen un formato bastante colorido, ya que se ha aplicado un coloreado basado en *mapas de calor*. Esta técnica colorea las celdas con un color más cálido —en este caso colores rojizos— cuanto mejor es el valor representado por dicha celda y un color más frío —en este caso colores azules— cuanto peor es el valor representado por dicha celda. En las tablas 7.1, 7.2 y 7.3 se ha aplicado por filas, pudiendo comparar así el comportamiento de cada algoritmo. En las tablas 7.4, 7.5, 7.6 y 7.7 se ha aplicado por columnas, permitiendo comparar los algoritmos y ver el mejor valor de cada métrica. En el apéndice 9 se encuentran las tablas con toda la información recogida, para que se pueda verificar que las tablas presentadas en esta sección son reales.

Todas las ejecuciones han sido realizadas con los valores por defecto de los parámetros. Dichos valores los podemos ver en la sección 6.2 o en la página web que contiene la documentación del proyecto, <https://nestorrv.github.io>.

Para comparar como de efectivo es el proceso de reducción se ha comparado el tamaño del conjunto de datos antes y después de la aplicación de los

algoritmos. Esta comparación la podemos ver en la tabla 7.1. Por lo general, el algoritmo que mejor se comporta es *IPADE-ID* [17], junto con *CPM* [10]. En la tabla 7.2 podemos ver los porcentajes de reducción para los distintos algoritmos. Si nos fijamos en los valores medios, podemos ver que se obtiene un alto porcentaje de reducción, en torno a la mitad del tamaño original.

En la tabla 7.3 podemos ver como se comporta el *Imbalanced Ratio* tras aplicar los distintos algoritmos. En este caso, el algoritmo que mejor se comporta —por lo general— es *IHTS* [16] junto con *ClusterOSS* [11]. Si nos fijamos en los valores medios, podemos ver que se obtiene un *Imbalanced Ratio* bastante menor que en el conjunto de datos original, lo cual nos indica que se ha conseguido una mejor proporción de instancias positivas con respecto a instancias negativas.

En las tablas 7.4, 7.5, 7.6 y 7.7 podemos ver los valores medios de las métricas comentadas en la sección 7.3 para las cinco ejecuciones —obtenidas al aplicar *k-fold*— de los conjuntos de datos antes y después de ser reducidos por los algoritmos. Como podemos ver, los valores medios obtenidos tras aplicar los algoritmos es bastante similar a la media de los conjuntos de datos original —los cuales podemos ver en la fila identificada por *original*—. En todos los casos, hay un algoritmo que mejora los valores del conjunto original.

Tal y como hemos visto en esta sección, realizar un proceso de *undersampling* aporta ventajas, pero tiene un inconveniente, y es que no es un proceso gratis. Este proceso lleva asociado un tiempo de ejecución. En la tabla 7.8 podemos ver los tiempos de ejecución de cada algoritmo en cada conjunto de datos. El formato de dicha tabla es: *minutos:segundos:milisegundos*. Como podemos ver, la mayoría de los algoritmos son bastante rápidos, excepto *EUS* [15] e *IPADE-ID*, ya que al ser algoritmos evolutivos necesitan más tiempo de ejecución.

Dataset	Tamaño Original										Tamaño Reducido										Media	
	BC	Cluster	OS	CNN	CPM	EE	ENN	EUS	IHTS	IPADE	NCL	NM	OSS	RU	SBC	TL						
ecoli-0_vs_1	220	154	64	17	18	154	220	154	106	19	210	154	127	154	211	210	131.46667	131.46667	131.46667	131.46667	131.46667	
ecoli1	336	154	56	80	85	154	321	154	88	37	299	154	226	154	291	296	169.93333	169.93333	169.93333	169.93333	169.93333	
ecoli2	336	104	34	67	63	104	329	104	63	24	317	104	281	104	309	312	154.6	154.6	154.6	154.6	154.6	
ecoli3	336	70	42	64	50	70	324	70	47	33	314	70	250	70	303	310	139.13333	139.13333	139.13333	139.13333	139.13333	
glass-0-1-2-3-vs-4-5-6	214	102	30	31	28	102	206	102	56	31	196	102	102	102	202	202	112.53333	112.53333	112.53333	112.53333	112.53333	
glass0	214	140	41	78	72	140	183	140	98	38	171	140	134	140	177	170	124.13333	124.13333	124.13333	124.13333	124.13333	
glass1	214	152	34	87	81	152	192	152	115	40	171	152	163	152	185	172	133.33333	133.33333	133.33333	133.33333	133.33333	
glass6	214	58	36	19	22	58	213	58	34	26	198	58	87	58	202	202	88.6	88.6	88.6	88.6	88.6	
haberman	306	162	55	163	147	162	273	163	82	38	235	162	201	162	209	212	161.73333	161.73333	161.73333	161.73333	161.73333	
iris0	150	100	13	2	2	100	150	100	68	13	150	100	49	100	148	146	82.73333	82.73333	82.73333	82.73333	82.73333	
new-thyroid1	215	70	41	11	13	70	213	70	68	24	202	70	61	70	208	205	93.06667	93.06667	93.06667	93.06667	93.06667	
new-thyroid2	215	70	36	15	16	70	213	70	40	23	203	70	93	70	206	205	93.33333	93.33333	93.33333	93.33333	93.33333	
page-blocks0	5472	1118	502	514	514	1118	5410	1117	665	135	5255	1118	4598	1118	3312	5186	2112					
pima	768	536	161	377	374	536	678	535	287	55	580	536	538	536	536	558	454.86667	454.86667	454.86667	454.86667	454.86667	
segment0	2308	658	287	69	65	658	2290	652	725	38	2277	658	1841	658	1830	2172	991.86667	991.86667	991.86667	991.86667	991.86667	
vehicle0	846	398	148	133	154	398	829	399	345	73	787	398	724	398	690	734	440.53333	440.53333	440.53333	440.53333	440.53333	
vehicle1	846	434	187	380	386	434	756	422	293	161	661	434	612	434	565	618	451.8	451.8	451.8	451.8	451.8	
vehicle2	846	436	161	138	143	436	816	433	353	53	783	436	729	436	616	736	447	447	447	447	447	
vehicle3	846	424	127	356	370	424	774	417	458	131	674	424	610	424	558	624	453	453	453	453	453	
wisconsin	683	478	28	74	73	478	672	478	421	52	662	478	303	478	681	663	401.26667	401.26667	401.26667	401.26667	401.26667	
yeast1	1484	858	294	681	696	858	1325	859	508	150	1146	858	1111	858	1481	1114	853.13333	853.13333	853.13333	853.13333	853.13333	
yeast3	1484	326	151	245	238	326	1451	326	179	92	1395	326	1362	326	1109	1362						

Tabla 7.1: Tamaño de los conjuntos de datos reducidos.

Dataset	BC	Cluster	OS	CNN	CPM	EE	ENN	EUS	IHTS	IPADE	NCL	NM	OSS	RU	SBC	TL	Media
ecoli-0_vs_1	30	70.91	92.27	91.82	30	0	30	51.82	91.36	4.55	30	42.27	30	4.09	4.55	40.24	
ecoli1	54.17	83.33	76.19	74.7	54.17	4.46	54.17	73.81	88.99	11.01	54.17	32.74	54.17	13.39	11.9	49.42	
ecoli2	69.05	89.88	80.06	81.25	69.05	2.08	69.05	81.25	92.86	5.65	69.05	16.37	69.05	8.04	7.14	53.99	
ecoli3	79.17	87.5	80.95	85.12	79.17	3.57	79.17	86.01	90.18	6.55	79.17	25.6	79.17	9.82	7.74	58.59	
glass-0-1-2-3_vs_4-5-6	52.34	85.98	85.51	86.92	52.34	3.74	52.34	73.83	85.51	8.41	52.34	5.61	5.61	5.61	47.41		
glass0	34.58	80.84	63.55	66.36	34.58	14.49	34.58	54.21	82.24	20.09	34.58	37.38	34.58	17.29	20.56	41.99	
glass1	28.97	84.11	59.35	62.15	28.97	10.28	28.97	46.26	81.31	20.09	28.97	23.83	28.97	13.55	19.63	37.69	
glass6	72.9	83.18	91.12	89.72	72.9	0.47	72.9	84.11	87.85	7.48	72.9	59.35	72.9	5.61	5.61	58.6	
haberman	47.06	82.03	46.73	51.96	47.06	10.78	46.73	73.2	87.58	23.2	47.06	34.31	47.06	31.7	30.72	47.15	
iris0	33.33	91.33	98.67	98.67	33.33	0	33.33	54.67	91.33	0	33.33	67.33	33.33	1.33	2.67	44.84	
new-thyroid1	67.44	80.93	94.88	93.95	67.44	0.93	67.44	68.37	88.84	6.05	67.44	71.63	67.44	3.26	4.65	56.71	
new-thyroid2	67.44	83.26	93.02	92.56	67.44	0.93	67.44	81.4	89.3	5.58	67.44	56.74	67.44	4.19	4.65	56.59	
page-blocks0	79.57	90.83	90.61	90.61	79.57	1.13	79.59	87.85	97.53	3.97	79.57	15.97	79.57	39.47	5.23	61.4	
pima	30.21	79.04	50.91	51.3	30.21	11.72	30.34	62.63	92.84	24.48	30.21	29.95	30.21	30.21	27.34	40.77	
segment0	71.49	87.56	97.01	97.18	71.49	0.78	71.75	68.59	98.35	1.34	71.49	20.23	71.49	20.71	5.89	57.02	
vehicle0	52.96	82.51	84.28	81.8	52.96	2.01	52.84	59.22	91.37	6.97	52.96	14.42	52.96	18.44	13.24	47.93	
vehicle1	48.7	77.9	55.08	54.37	48.7	10.64	50.12	65.37	80.97	21.87	48.7	27.66	48.7	33.22	26.95	46.6	
vehicle2	48.46	80.97	83.69	83.1	48.46	3.55	48.82	58.27	93.74	7.45	48.46	13.83	48.46	27.19	13	47.16	
vehicle3	49.88	84.99	57.92	56.26	49.88	8.51	50.71	45.86	84.52	20.33	49.88	27.9	49.88	34.04	26.24	46.45	
wisconsin	30.01	95.9	89.17	89.31	30.01	1.61	30.01	38.36	92.39	3.07	30.01	55.64	30.01	0.29	2.93	41.25	
yeast1	42.18	80.19	54.11	53.1	42.18	10.71	42.12	65.77	89.89	22.78	42.18	25.13	42.18	0.2	24.93	42.51	
yeast3	78.03	89.82	83.49	83.96	78.03	2.22	78.03	87.94	93.8	6	78.03	8.22	78.03	25.27	8.22	58.61	

Tabla 7.2: Porcentaje de reducción.

Dataset		IR Original	BC	ClusterOSS	CNN	CPM	EE	ENN	EUS	IHTS	IPADE	NCL	NM	OSS	RU	SBC	TL	Media
ecoli-0_vs_1	1.85714	1	0.30612	2	1	1.85714	1	0.37662	0.72727	1.72727	1	0.76339	1	1.7426	1.91667	1	1.16935	
ecoli1	3.36364	1	1.666667	1.02381	1	3.16883	1	0.14286	0.54167	2.98667	1	2.42424	1	2.77922	3.16901	1	1.60055	
ecoli2	5.46154	1	3.25	2.35	1	5.32692	1	0.21154	0.5	5.21569	1	4.73469	1	4.94231	5.93333	2.67802		
ecoli3	8.6	1	0.5	1.56	1	8.25714	1	0.34286	0.73684	8.23529	1	6.57576	1	7.65714	9.33333	3.29807		
glass-0-1-2-3_vs_4-5-6	3.19608	1	2	0.9375	1.15385	1	3.03922	1	0.09804	0.47619	2.92	1	3.17021	1	2.96078	3.3913	1.67647	
glass0	2.05714	1	1.92857	1.51613	1.48276	1	1.61429	1	0.4	1.11111	1.89831	1	1.57632	1	1.52857	1.83333	1.326	
glass1	1.81579	1	0.61905	1.175	1.25	1	1.52632	1	0.51316	0.666667	1.375	1	1.76271	1	1.43421	1.77419	1.13975	
glass6	6.37931	1	0.56522	2.16667	3.4	1	6.34483	1	0.17241	0.85714	5.82759	1	2.22222	1	5.96552	6.21429	2.58239	
haberman	2.77778	1	0.89655	1.50769	1.29688	1	2.37037	1.01235	0.01235	0.52	2.26389	1	2.65455	1	1.58025	2.65517	1.38467	
iris0	2	1	12	1	1	2	1	0.36	0.18182	2	1	0.02083	1	1.96	2.04167	1.83762		
new-thyroid1	5.14286	1	0.51852	1.2	0.85714	1	5.08571	1	0.94286	0.33333	4.94118	1	1.10345	1	4.94286	5.21212	2.00914	
new-thyroid2	5.14286	1	0.44	1.14286	0.77778	1	5.08571	1	0.14286	0.35294	4.97059	1	1.90625	1	4.88571	5.21212	1.99445	
page-blocks0	8.78891	1	0.0308	1.41315	1.45933	1	8.678	0.99821	0.18962	0.90141	8.51993	1	7.65913	1	4.92487	8.95333	3.18189	
pima	1.86567	1	0.14184	1.17919	1.14943	1	1.52985	0.99627	0.0709	0.77419	1.46809	1	1.69	1	1	1.87620	1.0584	
segment0	6.0152	1	0.05515	2.13636	2.42105	1	5.96049	0.98176	1.20365	0.72727	6.00615	1	5.05592	1	4.56231	5.96154	2.60478	
vehicle0	3.25126	1	0.0963	1.33333	1.02632	1	3.16583	1.00503	0.73367	0.78049	3.12042	1	3.02222	1	2.46734	3.44848	1.6133	
vehicle1	2.89862	1	1.28049	1.55034	1.57333	1	2.48387	0.9447	0.35023	1.29761	2.35533	1	2.77778	1	3.03922	1.54844		
vehicle2	2.88073	1	0.11034	1.875	1.34426	1	2.74312	0.98624	0.61927	0.60606	2.72857	1	2.83684	1	1.82569	2.78182	1.50192	
vehicle3	2.99057	1	0.64935	1.45517	1.46667	1	2.65094	0.96698	1.01638	1.01538	2.51042	1	1.74233	1	1.63208	2.78182	1.53543	
wisconsin	1.85774	1	0.64706	0.39623	0.40385	1	1.81172	1	0.76151	0.625	1.79325	1	0.35874	1	1.84937	1.84549	1.03281	
yeast1	2.45921	1	1.9697	1.42239	1.31229	1	2.08858	1.00233	0.18415	0.78571	2.07239	1	2.43963	1	2.45221	2.49216	1.48151	
yeast3	8.10429	1	0.10219	2.02469	1.8	1	7.90184	1	0.09816	1.2439	7.77358	1	8.2027	1	5.80368	8.01987	3.19804	

Tabla 7.3: *Imbalanced Ratio* de los conjuntos de datos reducidos.

ecoli												ecoli2											
Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%					
BC	0.979	0.961	95.944	0.976	0.976	98.167	0.877	0.869	85.404	0.873	0.871	85.68	0.873	0.866	87.61	0.892	0.889	86.654					
CNN	0.98	0.98	98.111	0.964	0.964	96.833	0.904	0.81	89.614	0.825	0.813	88.125	0.757	0.747	89.632	0.691	0.548	89.963					
CPM	0.983	0.982	98.611	0.961	0.96	97.278	0.832	0.765	86.324	0.629	0.421	82.224	0.755	0.763	86.415	0.63	0.378	85.527					
ClusterOSS	0.924	0.919	91	0.952	0.95	95.056	0.7	0.675	71.746	0.72	0.561	83.125	0.763	0.622	84.504	0.741	0.541	86.691					
EE	0.976	0.975	97.722	0.947	0.946	95.389	0.917	0.893	87.813	0.879	0.877	86.029	0.871	0.835	82.463	0.906	0.904	88.346					
ENN	0.983	0.982	98.611	0.953	0.951	96.778	0.906	0.88	90.147	0.845	0.843	87.224	0.881	0.886	94.982	0.779	0.748	89.945					
EUS	0.969	0.968	96.833	0.967	0.966	97.278	0.915	0.91	89.963	0.879	0.877	86.085	0.851	0.843	85.404	0.916	0.915	89.577					
IHTS	0.972	0.965	96.278	0.974	0.974	97.056	0.728	0.666	69.063	0.729	0.577	60.864	0.734	0.377	33.713	0.548	0.187	23.842					
IPADE-ID	0.968	0.917	91.278	0.607	0.365	48.722	0.78	0.703	86.691	0.5	0	71.114	0.763	0.656	77.574	0.584	0.184	86.324					
NCL	0.981	0.981	98.222	0.969	0.968	97.778	0.886	0.815	88.419	0.824	0.817	86.029	0.848	0.877	94.375	0.776	0.742	91.103					
NM	0.98	0.98	98.167	0.958	0.956	97.167	0.917	0.881	89.246	0.852	0.849	87.996	0.805	0.767	75.809	0.825	0.827	77.555					
OSS	0.984	0.98	98.167	0.962	0.961	96.889	0.861	0.826	89.89	0.815	0.805	88.401	0.841	0.838	93.474	0.763	0.724	91.985					
RU	0.958	0.957	95.5	0.948	0.946	95.944	0.909	0.882	86.305	0.885	0.883	86.305	0.847	0.845	83.438	0.805	0.894	86.563					
SCB	0.97	0.965	96.278	0.967	0.967	97.722	0.907	0.89	88.474	0.881	0.88	87.279	0.877	0.826	93.474	0.799	0.775	91.415					
TL	0.984	0.984	98.611	0.955	0.953	96.833	0.883	0.834	89.559	0.82	0.801	89.522	0.784	0.784	92.904	0.786	0.758	92.316					
original	0.979	0.975	97.722	0.967	0.967	97.722	0.876	0.819	89.228	0.794	0.773	87.702	0.867	0.856	94.393	0.777	0.739	91.415					
ecoli3												glass0											
Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%					
BC	0.768	0.751	79.485	0.877	0.866	81.287	0.905	0.892	91.077	0.885	0.883	90.55	0.79	0.78	77.065	0.712	0.649	61.196					
CNN	0.653	0.512	91.085	0.5	0	89.577	0.811	0.788	79.402	0.807	0.786	88.947	0.579	0.378	64.507	0.5	0	67.287					
CPM	0.657	0.522	64.007	0.557	0.274	88.401	0.825	0.803	79.569	0.8	0.776	79.975	0.677	0.665	68.715	0.5	0	67.287					
ClusterOSS	0.807	0.757	74.522	0.782	0.678	81.232	0.857	0.827	86.077	0.839	0.822	90.694	0.692	0.667	68.217	0.581	0.506	67.763					
EE	0.875	0.858	81.985	0.869	0.866	81.121	0.919	0.901	90.478	0.88	0.876	91.388	0.794	0.77	78.018	0.701	0.637	60.288					
ENN	0.823	0.765	92.5	0.661	0.565	91.36	0.94	0.932	94.928	0.845	0.833	89.785	0.789	0.779	78.04	0.751	0.735	71.008					
EUS	0.84	0.836	83.272	0.884	0.879	81.544	0.878	0.871	87.823	0.863	0.857	90.5	0.846	0.846	78.04	0.695	0.627	59.338					
IHTS	0.59	0.408	49.118	0.697	0.495	54.704	0.778	0.774	77.153	0.803	0.699	77.321	0.705	0.639	63.898	0.641	0.481	52.248					
IPADE-ID	0.908	0.857	87.61	0.715	0.581	71.581	0.817	0.629	83.876	0.5	0	46.316	0.625	0.246	63.566	0.5	0	53.643					
NCL	0.803	0.715	92.555	0.514	0.076	89.871	0.871	0.894	93.421	0.886	0.882	92.656	0.775	0.722	76.6	0.647	0.603	69.169					
NM	0.706	0.61	74.853	0.643	0.636	80.656	0.907	0.903	91.675	0.885	0.884	90.694	0.749	0.702	69.192	0.47	0.42	51.406					
OS	0.8	0.707	91.415	0.509	0.074	88.989	0.873	0.866	91.148	0.865	0.858	88.206	0.779	0.786	79.402	0.568	0.348	70.554					
RU	0.823	0.811	80.68	0.863	0.858	80.074	0.939	0.918	92.057	0.879	0.876	90.622	0.827	0.783	76.622	0.712	0.65	61.24					
SCB	0.862	0.778	89.908	0.818	0.805	92.279	0.926	0.919	93.493	0.867	0.86	91.675	0.777	0.755	75.692	0.695	0.644	63.998					
TL	0.82	0.769	91.985	0.5	0	89.577	0.919	0.915	93.947	0.893	0.884	92.967	0.829	0.803	81.805	0.514	0.089	67.763					
original	0.805	0.753	93.438	0.5	0	89.577	0.909	0.886	92.057	0.861	0.853	91.603	0.791	0.751	77.065	0.55	0.307	69.612					

Tabla 7.4: Métricas clasificación (parte 1).

glass6												haberman											
Algoritmo	j48auc	j48gm	j48-%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48-%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48-%	SVMauc	SVMgn	SVM-%					
BC	0.69	0.651	68.612	0.602	0.525	51.89	0.874	0.869	85.061	0.873	0.864	90.19	0.625	0.615	70.612	0.615	0.604	70.656					
CNN	0.5	0	59.139	0.53	0.32	61.053	0.812	0.707	58.804	0.708	0.506	89.247	0.567	0.366	66.485	0.498	0	73.259					
CPM	0.573	0.4	55.407	0.504	0.195	57.177	0.773	0.721	70.21	0.547	0.196	87.386	0.54	0.206	73.582	0.496	0	72.937					
ClusterOSS	0.557	0.524	53.636	0.511	0.31	45.478	0.907	0.895	89.723	0.913	0.908	89.623	0.579	0.552	60.69	0.508	0.089	43.515					
EE	0.727	0.702	72.789	0.58	0.482	48.565	0.924	0.924	90.133	0.908	0.906	94.385	0.639	0.63	67.108	0.508	0.513	73.281					
ENN	0.692	0.644	72.249	0.487	0.66	62.177	0.91	0.898	93.444	0.916	0.912	95.77	0.62	0.575	69.922	0.554	0.368	74.527					
EUS	0.733	0.71	70.718	0.603	0.515	51.435	0.901	0.89	89.269	0.898	0.894	92.071	0.622	0.607	71.068	0.615	0.513	75.284					
IHTS	0.623	0.407	49.234	0.511	0.065	36.77	0.846	0.832	78.815	0.758	0.647	65.183	0.504	0.236	33.515	0.5	0	26.418					
IPADE-ID	0.545	0.233	52.775	0.5	0	53.684	0.933	0.253	23.344	0.578	0.18	59.003	0.645	0.332	45.94	0.5	0	26.418					
NCL	0.691	0.673	71.627	0.504	0.119	64.139	0.884	0.891	93.92	0.886	0.879	94.862	0.589	0.54	69.689	0.503	0.055	73.281					
NM	0.678	0.653	64.115	0.479	0.451	51.124	0.877	0.867	86.013	0.936	0.933	96.268	0.618	0.589	64.627	0.625	0.59	71.101					
OSS	0.757	0.733	75.12	0.54	0.14	65.957	0.787	0.847	88.793	0.885	0.876	95.781	0.597	0.358	74.271	0.408	0	73.237					
RU	0.719	0.72	70.957	0.583	0.494	49.545	0.896	0.897	89.767	0.914	0.909	94.839	0.608	0.581	66.051	0.505	0.529	63.448					
SCB	0.722	0.713	72.392	0.54	0.385	59.091	0.912	0.903	93.942	0.903	0.895	95.803	0.628	0.615	70.845	0.529	0.223	73.56					
TL	0.713	0.717	75.718	0.5	0	64.593	0.829	0.865	92.99	0.886	0.878	95.327	0.583	0.45	71.947	0.5	0	73.582					
original	0.721	0.664	73.23	0.5	0	64.593	0.869	0.833	93.001	0.889	0.882	95.792	0.598	0.571	70.289	0.5	0	73.582					
iris0												new-thyroid1											
Algoritmo	j48auc	j48gm	j48-%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48-%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48-%	SVMauc	SVMgn	SVM-%					
BC	0.99	0.99	99.333	1	1	100	0.951	0.95	95.814	0.96	0.959	97.209	0.921	0.919	92.558	0.966	0.965	98.14					
CNN	0.5	0	33.333	0.99	0.99	99.333	0.898	0.898	90.698	0.885	0.87	88.372	0.784	0.76	86.977	0.961	0.96	93.488					
CPM	0.5	0	33.333	0.87	0.846	88.667	0.825	0.823	84.186	0.811	0.736	68.372	0.864	0.852	90.698	0.98	0.98	98.605					
ClusterOSS	0.69	0.464	79.333	0.91	0.894	94	0.955	0.954	94.419	0.954	0.953	91.628	0.902	0.897	89.302	0.96	0.959	97.209					
EE	0.99	0.99	99.333	1	1	100	0.924	0.923	93.023	0.954	0.953	98.14	0.927	0.915	91.628	0.969	0.968	98.605					
ENN	0.98	0.979	98.667	1	1	100	0.936	0.914	96.279	0.786	0.754	93.023	0.9	0.864	94.884	0.757	0.707	92.093					
EUS	0.98	0.979	98.667	1	1	100	0.924	0.923	93.023	0.943	0.94	96.279	0.91	0.908	88.837	0.94	0.934	97.674					
IHTS	0.99	0.99	99.333	1	1	100	0.927	0.926	91.628	0.88	0.863	81.86	0.811	0.696	68.372	0.686	0.595	47.442					
IPADE-ID	0.56	0.155	70.667	0.56	0.205	70.667	0.64	0.434	59.07	0.69	0.613	82.791	0.6	0.292	44.651	0.665	0.438	76.744					
NCL	0.99	0.99	99.333	1	1	100	0.95	0.949	95.349	0.786	0.732	93.023	0.929	0.942	96.279	0.771	0.725	92.558					
NM	0.99	0.99	99.333	1	1	100	0.976	0.95	97.674	0.977	0.977	98.14	0.969	0.954	92.279	0.983	0.982	99.07					
OSS	0.7	0.4	60	0.995	0.995	99.333	0.941	0.9	95.349	0.966	0.965	98.14	0.948	0.943	96.744	0.871	0.851	95.814					
RU	0.99	0.99	99.333	1	1	100	0.941	0.94	93.953	0.954	0.953	98.14	0.91	0.905	92.558	0.954	0.951	98.14					
SCB	0.98	0.979	98.667	1	1	100	0.888	0.897	94.884	0.757	0.715	92.093	0.9	0.888	94.884	0.771	0.743	93.023					
TL	0.98	0.979	98.667	1	1	100	0.862	0.901	94.884	0.786	0.738	93.023	0.94	0.934	94.884	0.771	0.729	92.558					

Tabla 7.5: Métricas clasificación (parte 2).

Algoritmo	page-blocks0						pina						segment0					
	j48auc	j48gm	j48_%	SVMauc	SVMgmn	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgmn	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgmn	SVM_%
BC	0.954	0.942	93.422	0.85	0.847	90.423	0.737	0.697	68.498	0.743	0.742	75.246	0.978	0.978	0.978	0.978	0.978	97.268
CNN	0.915	0.901	94.681	0.628	0.487	91.868	0.694	0.623	70.813	0.694	0.663	75.519	0.943	0.947	96.142	0.674	0.452	90.291
CPM	0.933	0.693	93.386	0.673	0.577	92.47	0.679	0.605	70.981	0.71	0.683	76.569	0.899	0.889	93.586	0.786	0.633	93.667
ClusterOSS	0.742	0.693	54.418	0.5	0	10.216	0.578	0.561	55.219	0.574	0.423	56.15	0.857	0.757	70.982	0.817	0.79	70.302
EE	0.963	0.948	94.407	0.849	0.845	90.607	0.713	0.7	69.658	0.739	0.738	74.872	0.982	0.983	98.181	0.98	0.979	97.575
ENN	0.951	0.928	97.241	0.743	0.701	93.787	0.755	0.724	73.688	0.721	0.714	74.865	0.985	0.983	99.35	0.99	0.99	99.553
EUS	0.948	0.939	93.677	0.855	0.852	90.808	0.726	0.677	68.204	0.738	0.735	74.728	0.978	0.979	98.181	0.979	0.978	97.618
IHTS	0.708	0.43	40.435	0.565	0.163	26.727	0.571	0.229	43.413	0.522	0.105	38.315	0.875	0.804	72.391	0.882	0.869	80.276
IPADE-ID	0.5	0	10.425	0.5	0	10.216	0.5	0.082	35.154	0.5	0	34.894	0.538	0.141	87.174	0.597	0.454	76.395
NCL	0.926	0.914	96.783	0.733	0.686	93.805	0.714	0.71	71.222	0.738	0.731	76.688	0.983	0.981	98.959	0.99	0.99	99.653
NM	0.796	0.775	68.139	0.739	0.737	69.573	0.737	0.709	70.713	0.735	0.734	73.831	0.95	0.948	92.117	0.952	0.952	92.891
OSS	0.942	0.911	96.838	0.716	0.657	93.786	0.73	0.697	74.211	0.719	0.697	77.08	0.983	0.984	98.22	0.992	0.992	99.697
RU	0.952	0.942	93.53	0.85	0.847	90.241	0.721	0.688	68.884	0.734	0.732	74.737	0.977	0.976	98.137	0.979	0.979	97.441
SCB	0.959	0.946	94.736	0.802	0.785	93.384	0.725	0.702	68.638	0.752	0.75	74.222	0.98	0.976	98.85	0.989	0.989	99.263
TL	0.952	0.922	57.113	0.701	0.632	93.476	0.687	0.649	70.176	0.712	0.687	76.816	0.986	0.986	99.394	0.99	0.99	99.653
original	0.928	0.91	96.82	0.699	0.631	93.458	0.75	0.715	74.233	0.712	0.687	76.83	0.983	0.983	99.133	0.99	0.99	99.653
vehicle0																		
Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgmn	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgmn	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgmn	SVM_%
BC	0.938	0.931	92.904	0.95	0.949	92.678	0.731	0.718	71.508	0.713	0.712	70.346	0.951	0.943	93.619	0.906	0.904	88.666
CNN	0.878	0.86	90.666	0.896	0.896	93.972	0.664	0.534	73.525	0.511	0.09	74.825	0.889	0.868	90.089	0.657	0.548	81.895
CPM	0.939	0.903	92.322	0.93	0.929	95.157	0.685	0.569	73.725	0.542	0.21	76.142	0.907	0.894	91.605	0.716	0.656	84.285
ClusterOSS	0.862	0.814	78.495	0.739	0.71	63.569	0.552	0.499	52.218	0.6	0.491	63.11	0.819	0.797	77.826	0.625	0.479	50.257
EE	0.92	0.908	91.619	0.952	0.951	93.151	0.768	0.737	70.693	0.729	0.726	71.789	0.947	0.933	94.101	0.908	0.907	88.876
ENN	0.945	0.918	93.478	0.943	0.943	94.79	0.725	0.714	73.41	0.667	0.615	79.072	0.949	0.936	95.042	0.902	0.898	93.159
EUS	0.945	0.922	92.21	0.948	0.947	92.325	0.744	0.731	72.808	0.727	0.726	71.97	0.956	0.949	94.675	0.91	0.91	89.352
IHTS	0.83	0.769	73.473	0.783	0.701	66.87	0.652	0.545	56.662	0.627	0.439	50.22	0.806	0.753	72.591	0.743	0.542	62.943
IPADE-ID	0.5	0	65.889	0.5	0	76.478	0.5	0	44.94	0.521	0.098	68.237	0.5	0	54.94	0.502	0.025	26.001
NCL	0.932	0.912	93.145	0.939	0.938	95.157	0.687	0.647	74.242	0.617	0.499	79.091	0.96	0.952	95.863	0.888	0.884	92.554
NM	0.746	0.777	73.023	0.956	0.956	95.986	0.673	0.599	63.236	0.708	0.706	71.15	0.937	0.944	94.913	0.902	0.901	90.072
OS	0.935	0.898	93.14	0.944	0.943	95.734	0.712	0.638	76.004	0.516	0.156	75.181	0.945	0.936	95.148	0.876	0.876	91.717
RU	0.93	0.906	90.792	0.955	0.954	93.366	0.752	0.732	71.881	0.712	0.712	70.464	0.929	0.926	92.084	0.914	0.914	89.952
SCB	0.933	0.93	91.722	0.96	0.959	94.081	0.726	0.733	71.976	0.714	0.708	75.301	0.945	0.94	95.154	0.913	0.912	92.439
TL	0.95	0.916	93.389	0.933	0.932	95.266	0.729	0.675	75.408	0.538	0.215	76.128	0.948	0.942	95.991	0.882	0.877	92.31
original	0.938	0.928	94.678	0.94	0.939	95.863	0.719	0.638	73.864	0.531	0.246	75.769	0.967	0.951	95.986	0.89	0.887	92.563

Tabla 7.6: Métricas clasificación (parte 3).

Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48_%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48_%	SVMauc	SVMgn	SVM-%								
	vehicle									wiscosin									yeast1							
BC	0.761	0.731	72.561	0.723	0.719	0.719	69.984	0.933	0.951	95.157	0.968	0.968	96.771	0.722	0.686	68.533	0.717	0.715	69.944							
CNN	0.664	0.553	73.772	0.523	0.142	75.887	0.928	0.908	90.32	0.969	0.969	96.923	0.707	0.664	70.351	0.544	0.307	72.845								
CPM	0.67	0.592	68.054	0.506	0.073	75.181	0.937	0.888	89.18	0.947	0.946	94.716	0.715	0.621	74.46	0.573	0.405	74.056								
ClusterOSS	0.608	0.588	58.499	0.593	0.556	56.421	0.5	0	34.992	0.5	0	34.992	0.584	0.468	49.327	0.532	0.239	50.337								
EE	0.765	0.748	73.155	0.716	0.714	69.629	0.935	0.932	93.705	0.973	0.973	97.224	0.741	0.714	71.699	0.71	0.708	69.068								
ENN	0.737	0.692	74.564	0.535	0.245	74.099	0.961	0.952	95.464	0.972	0.972	97.074	0.732	0.665	74.73	0.642	0.578	76.011								
EUS	0.725	0.721	70.811	0.709	0.708	69.273	0.959	0.948	94.734	0.97	0.969	96.928	0.729	0.706	72.105	0.706	0.706	69.542								
IHTS	0.676	0.555	53.106	0.626	0.437	48.231	0.953	0.938	93.54	0.973	0.972	97.072	0.544	0.257	35.106	0.518	0.106	31.602								
IPADE-ID	0.5	0	74.945	0.591	0.507	57.531	0.5	0	34.992	0.5	0	34.992	0.607	0.538	57.08	0.628	0.485	59.716								
NCL	0.758	0.689	73.889	0.519	0.126	74.943	0.946	0.938	94.588	0.973	0.973	97.215	0.745	0.656	74.596	0.608	0.505	74.932								
NM	0.66	0.601	61.317	0.688	0.684	72.108	0.941	0.937	94.278	0.971	0.971	97.076	0.672	0.659	68.462	0.633	0.633	71.966								
OSS	0.725	0.642	75.298	0.5	0	74.945	0.944	0.949	95.309	0.964	0.964	96.632	0.69	0.618	74.196	0.57	0.398	74.126								
RU	0.74	0.712	68.685	0.728	0.725	70.318	0.93	0.935	93.856	0.973	0.972	97.069	0.719	0.673	67.724	0.707	0.706	69.137								
SCB	0.74	0.714	71.488	0.65	0.548	72.088	0.962	0.952	95.461	0.968	0.967	96.926	0.723	0.692	68.667	0.703	0.697	67.051								
TL	0.677	0.596	73.517	0.5	0	74.945	0.952	0.938	94.734	0.968	0.967	97.069	0.711	0.63	73.049	0.57	0.4	73.991								
original	<b>0.747</b>	<b>0.667</b>	<b>76.604</b>	<b>0.5</b>	<b>0</b>	<b>74.945</b>	<b>0.944</b>	<b>0.94</b>	<b>94.727</b>	<b>0.969</b>	<b>0.968</b>	<b>97.067</b>	<b>0.73</b>	<b>0.59</b>	<b>74.663</b>	<b>0.575</b>	<b>0.416</b>	<b>74.125</b>								
yeast3																										
Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48_%	SVMauc	SVMgn	SVM-%	j48auc	j48gm	j48_%	SVMauc	SVMgn	SVM-%								
BC	0.906	0.913	90.361	0.902	0.901	90.157																				
CNN	0.914	0.922	91.95	0.692	0.619	92.117																				
CPM	0.91	0.893	94.009	0.779	0.751	93.732																				
ClusterOSS	0.882	0.851	77.456	0.612	0.443	30.983																				
EE	0.923	0.926	93.043	0.895	0.895	89.402																				
ENN	0.877	0.864	94.001	0.778	0.745	94.001																				
EUS	0.928	0.915	90.437	0.91	0.909	90.116																				
IHTS	0.743	0.596	50.119	0.678	0.491	47.699																				
IPADE-ID	0.694	0.512	90.912	0.56	0.261	89.823																				
NCL	0.89	0.846	93.806	0.744	0.701	93.538																				
NM	0.811	0.822	83.599	0.877	0.875	90.835																				
OSS	0.878	0.868	94.812	0.721	0.664	93.268																				
RU	0.92	0.913	91.254	0.897	0.897	89.294																				
SCB	0.922	0.881	93.734	0.796	0.768	94.339																				
TL	0.923	0.869	93.875	0.702	0.634	92.99																				
original	<b>0.897</b>	<b>0.876</b>	<b>94.546</b>	<b>0.734</b>	<b>0.687</b>	<b>93.392</b>																				

Tabla 7.7: Métricas clasificación (parte 4).

	ecoli-0_vs_1	ecoli1	ecoli2	ecoli3	glasso-0-1-2-3_vs_4-5-6	glass0	glass1	glass6	haberman	iris0	new-thyroid1
<b>BC</b>	000:000:728	000:000:943	000:000:058	000:000:095	000:000:063	000:000:013	000:000:012	000:000:015	000:000:026	000:000:006	000:000:027
<b>ClusterOSS</b>	000:000:771	000:000:167	000:000:139	000:000:167	000:000:151	000:000:152	000:000:062	000:000:107	000:000:167	000:000:024	000:000:061
<b>CNN</b>	000:000:084	000:000:071	000:000:064	000:000:078	000:000:023	000:000:046	000:000:025	000:000:045	000:000:082	000:000:005	000:000:010
<b>CPM</b>	000:000:084	000:000:020	000:000:033	000:000:044	000:000:036	000:000:010	000:000:010	000:000:008	000:000:017	000:000:002	000:000:009
<b>EE</b>	000:000:015	000:000:001	000:000:000	000:000:000	000:000:000	000:000:002	000:000:000	000:000:001	000:000:001	000:000:000	000:000:000
<b>ENN</b>	000:000:042	000:000:034	000:000:068	000:000:036	000:000:050	000:000:042	000:000:025	000:000:056	000:000:035	000:000:004	000:000:012
<b>EUS</b>	000:002:918	000:003:586	000:003:295	000:002:358	000:001:498	000:002:030	000:002:442	000:001:339	000:003:083	000:001:010	000:001:377
<b>IHTS</b>	000:000:312	000:000:007	000:000:004	000:000:004	000:000:006	000:000:004	000:000:004	000:000:002	000:000:002	000:000:001	000:000:007
<b>IPADE-ID</b>	000:004:243	000:008:113	000:005:634	000:007:009	000:006:745	000:006:703	000:008:332	000:004:476	000:008:927	000:001:604	000:004:143
<b>NCL</b>	000:000:040	000:000:048	000:000:058	000:000:046	000:000:023	000:000:020	000:000:020	000:000:026	000:000:025	000:000:007	000:000:017
<b>NM</b>	000:000:020	000:000:020	000:000:014	000:000:013	000:000:009	000:000:007	000:000:009	000:000:007	000:000:011	000:000:005	000:000:005
<b>OSS</b>	000:000:028	000:000:033	000:000:045	000:000:056	000:000:021	000:000:017	000:000:019	000:000:011	000:000:017	000:000:004	000:000:007
<b>RU</b>	000:000:001	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000
<b>SBC</b>	000:000:083	000:000:091	000:000:035	000:000:052	000:000:021	000:000:018	000:000:024	000:000:019	000:000:024	000:000:007	000:000:017
<b>TL</b>	000:000:012	000:000:017	000:000:015	000:000:014	000:000:007	000:000:010	000:000:011	000:000:007	000:000:006	000:000:003	000:000:005
	new-thyroid2	page-blocks0	pima	segment0	vehicle0	vehicle1	vehicle2	vehicle3	wisconsin	yeast1	yeast3
<b>BC</b>	000:000:014	000:004:347	000:000:104	000:001:894	000:000:239	000:000:183	000:000:223	000:000:276	000:000:073	000:000:517	000:000:347
<b>ClusterOSS</b>	000:000:117	000:011:931	000:000:433	000:007:765	000:000:848	000:000:998	000:000:237	000:000:527	000:000:149	000:000:855	000:000:738
<b>CNN</b>	000:000:011	000:010:274	000:000:282	000:003:018	000:000:304	000:000:369	000:000:388	000:000:394	000:000:140	000:001:061	000:000:775
<b>CPM</b>	000:000:004	000:003:708	000:000:097	000:001:556	000:000:179	000:000:237	000:000:156	000:000:145	000:000:042	000:000:286	000:000:194
<b>EE</b>	000:000:000	000:000:004	000:000:000	000:000:001	000:000:000	000:000:000	000:000:000	000:000:000	000:000:001	000:000:000	000:000:000
<b>ENN</b>	000:000:012	000:010:225	000:000:156	000:002:584	000:000:298	000:000:212	000:000:218	000:000:193	000:000:104	000:000:528	000:000:670
<b>EUS</b>	000:001:371	012:041:589	000:026:452	002:023:606	000:023:614	000:022:608	000:023:058	000:023:014	000:018:434	001:023:829	000:049:195
<b>IHTS</b>	000:000:003	000:000:316	000:000:017	000:001:112	000:000:028	000:000:046	000:000:027	000:000:048	000:000:009	000:000:033	000:000:022
<b>IPADE-ID</b>	000:002:805	011:046:999	000:022:285	001:012:644	000:052:309	002:004:026	000:027:480	001:03:100	000:027:857	004:010:420	002:027:460
<b>NCL</b>	000:000:022	000:013:062	000:000:206	000:003:946	000:000:372	000:000:396	000:000:400	000:000:360	000:000:157	000:000:723	000:000:869
<b>NM</b>	000:000:006	000:004:301	000:000:080	000:001:645	000:000:193	000:000:195	000:000:152	000:000:212	000:000:067	000:000:342	000:000:244
<b>OSS</b>	000:000:014	000:006:900	000:000:146	000:002:768	000:000:299	000:000:314	000:000:323	000:000:320	000:000:103	000:000:597	000:000:468
<b>RU</b>	000:000:000	000:000:001	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000	000:000:000
<b>SBC</b>	000:000:041	000:012:108	000:000:212	000:001:538	000:000:269	000:000:226	000:000:212	000:000:292	000:000:018	000:000:065	000:000:389
<b>TL</b>	000:000:006	000:003:269	000:000:054	000:001:590	000:000:132	000:000:181	000:000:127	000:000:110	000:000:079	000:000:251	000:000:175

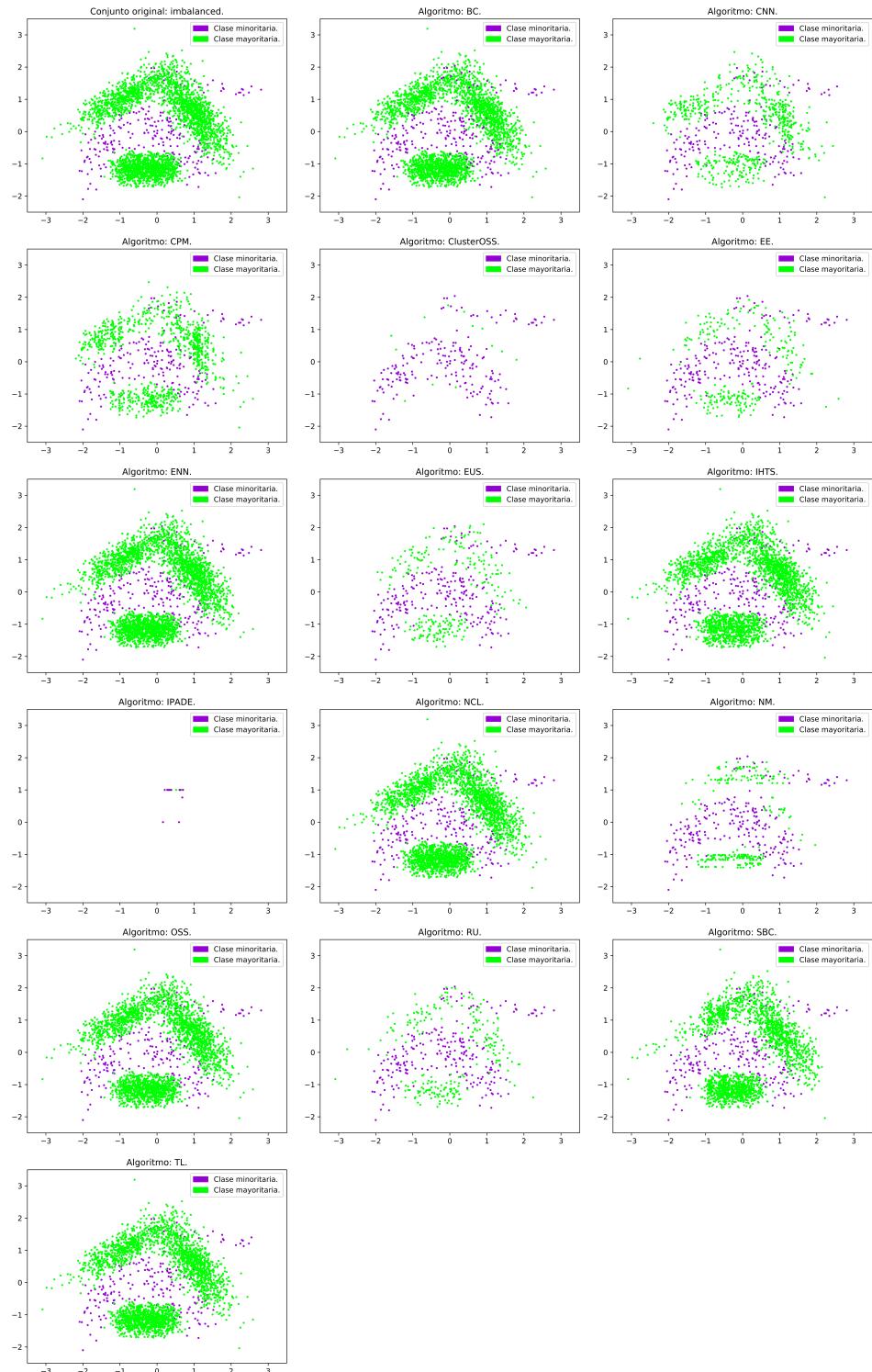
Tabla 7.8: Tiempos de ejecución.

## 7.5. Gráficos.

Hasta ahora solo hemos podido ver resultados numéricos de como se comportan los conjuntos de datos al ser reducidos, pero también es interesante ver como se comporta la nube de puntos del conjunto de datos obtenido con cada algoritmo. Para realizar esta tarea he cogido el conjunto de datos *banana.dat* [38] de *KEEL* [32] [33], ya que es un conjunto bidimensional y tiene una forma bastante característica cuando se representa de forma gráfica. Para adaptarlo al problema que nos ocupa —el problema de clasificación no balanceada— se ha reducido la clase minoritaria al diez por ciento de su tamaño, aumentando así el desbalanceo del conjunto de datos. Tras esto, se han aplicado los quince algoritmos sobre este conjunto desbalanceado.

En la figura 7.2 podemos ver una representación en dos dimensiones del conjunto desbalanceado —en la primera fila, primera columna— y la nube de puntos obtenida por cada uno de los algoritmos de esta biblioteca. Estas gráficas han sido creadas con [35] y el paquete *matplotlib* [39]. Como ya habíamos comentado en la sección 7.4 el algoritmo que más reduce es *IPADE-ID* [17] y así lo podemos ver en dicha imagen. Si nos fijamos en el comportamiento del algoritmo *ClusterOSS* [11] podemos ver que aumenta la frontera que hay entre ambas clases, reduciendo así el solapamiento de las mismas. Sin embargo, elimina bastantes puntos de la clase mayoritaria, generando así un conjunto desbalanceado pero sobre la que era la clase mayoritaria.

Los algoritmos *CNN* [13], *EE* [9], *EUS* [15] y *RU* —aunque este último lo hace de forma aleatoria— reducen el solapamiento dentro de la clase mayoritaria, teniendo así una nube de puntos que cubre el mismo espacio pero mucho menos densa, es decir, tenemos la misma representación espacial que el conjunto original pero con menos cantidad de datos. Como se puede apreciar en los gráficos de estos algoritmos, el conjunto de datos obtenido es un conjunto balanceado, al contrario que sucedía con *ClusterOSS*.

Figura 7.2: Nubes de puntos para *banana*.

## Capítulo 8

# Conclusiones y trabajos futuros.

En este proyecto se ha implementado una biblioteca con quince algoritmos de *undersampling*. En el momento de publicación del mismo no existe ninguna biblioteca con tantos algoritmos de *undersampling* y mucho menos en un lenguaje tan novedoso y a la orden del día como puede ser *Scala*.

En cuanto a nivel técnico, hemos visto la importancia de aplicar técnicas de *undersampling* antes de entrenar un clasificador por dos razones; primero, reducimos el número de elementos, lo cual nos permite realizar el proceso de entrenamiento más rápido y, segundo, en la mayoría de los casos obtendremos un resultado mejor —o similar— al resultado que hubiésemos obtenido con el conjunto de datos original.

### 8.1. Trabajos futuros.

Como trabajo futuro se podrían seguir implementando más algoritmos de *undersampling* para hacer la biblioteca más completa aún.

Otra mejora posible sería rehacer los algoritmos para optimizarlos con el conocimiento adquirido a lo largo de estos meses de aprendizaje en *Scala*, creando así versiones más eficientes y competitivas.

Finalmente, dado que entramos en una era de cantidades masivas de datos, se podrían adaptar los algoritmos para hacer un mayor uso del paralelismo, aunque la mayoría de algoritmos —no todos porque no todos pueden serlo— son paralelos. También se podría hacer uso de *Spark* [40] [41], un *framework* que permite introducir paralelismo en distintos lenguajes, entre ellos *Scala*. También se podría hacer uso de frameworks que permitan hacer ejecuciones en la *GPU*, reduciendo así el tiempo de ejecución.



# **Capítulo 9**

# **Apéndice.**

Recopilación de tablas que, por razones de espacio, no se han incluido en sus correspondientes capítulos pero son relevantes para verificar que las mostradas en los capítulos anteriores son reales.

ecoli-0_vs_1										ecoli1									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	1	0.983	97.778	0.969	0.968	97.778	0.985	0.959	97.059	0.889	0.889	89.706							
1	1	1	100	1	1	100	0.703	0.805	88.235	0.897	0.893	94.118							
2	0.969	0.968	97.778	0.938	0.935	95.556	0.895	0.728	85.294	0.762	0.735	86.765							
3	0.966	0.965	95.556	0.969	0.968	97.778	0.97	0.821	91.176	0.75	0.707	88.235							
4	0.962	0.961	97.5	0.962	0.961	97.5	0.828	0.782	84.375	0.672	0.638	79.688							
ecoli2										ecoli3									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.817	0.845	94.118	0.855	0.845	94.118	0.883	0.845	97.059	0.5	0	89.706							
1	0.765	0.776	89.706	0.692	0.65	85.294	0.924	0.743	92.647	0.5	0	89.706							
2	0.991	0.991	98.529	0.892	0.889	94.118	0.912	0.903	94.118	0.5	0	89.706							
3	0.769	0.674	91.176	0.636	0.522	88.235	0.644	0.737	91.176	0.5	0	89.706							
4	0.992	0.991	98.438	0.813	0.791	95.313	0.662	0.535	92.188	0.5	0	89.063							
glass-0-1-2-3_vs_4-5-6										glass0									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.994	0.891	93.182	0.848	0.84	90.909	0.867	0.714	74.419	0.518	0.263	67.442							
1	0.891	0.877	90.909	0.848	0.84	90.909	0.719	0.673	72.093	0.643	0.535	76.744							
2	0.826	0.905	95.455	0.818	0.798	90.909	0.836	0.695	81.395	0.554	0.371	69.767							
3	0.941	0.862	88.636	0.894	0.891	93.182	0.881	0.895	88.372	0.5	0	67.442							
4	0.892	0.895	92.105	0.896	0.895	92.105	0.654	0.678	69.048	0.536	0.364	66.667							
glass1										glass6									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.855	0.85	88.636	0.5	0	63.636	0.876	0.875	90.698	0.89	0.888	93.023							
1	0.748	0.701	72.727	0.5	0	63.636	0.833	0.707	93.023	0.833	0.816	95.349							
2	0.737	0.685	70.455	0.5	0	63.636	0.667	0.794	90.698	0.917	0.913	97.674							
3	0.674	0.555	65.909	0.5	0	63.636	0.995	0.816	95.349	0.917	0.913	97.674							
4	0.59	0.531	68.421	0.5	0	68.421	0.973	0.973	95.238	0.886	0.882	95.238							
haberman										iris0									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.571	0.605	69.355	0.5	0	72.581	1	1	100	1	1	100							
1	0.592	0.596	67.742	0.5	0	72.581	1	1	100	1	1	100							
2	0.62	0.539	69.355	0.5	0	72.581	0.95	0.949	96.667	1	1	100							
3	0.624	0.553	72.581	0.5	0	72.581	1	1	100	1	1	100							
4	0.585	0.562	72.414	0.5	0	77.586	1	1	100	1	1	100							
new-thyroid1										new-thyroid2									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.827	0.886	90.698	0.857	0.845	95.349	0.992	0.986	97.674	0.714	0.655	90.698							
1	1	100	1	1	100	1	0.909	0.9	93.023	0.786	0.756	93.023							
2	0.986	0.986	97.674	0.643	0.535	88.372	0.982	0.972	95.349	0.929	0.926	97.674							
3	0.508	0.645	88.372	0.714	0.655	90.698	0.887	0.886	90.698	0.714	0.655	90.698							
4	0.986	0.986	97.674	0.714	0.655	90.698	0.929	0.926	97.674	0.714	0.655	90.698							
page-blocks0										pima									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.966	0.904	96.895	0.708	0.65	92.968	0.75	0.716	73.377	0.709	0.687	75.974							
1	0.874	0.897	96.256	0.643	0.535	92.694	0.746	0.711	77.273	0.734	0.707	79.221							
2	0.925	0.909	96.895	0.694	0.626	93.425	0.729	0.697	69.481	0.737	0.719	78.571							
3	0.955	0.924	97.26	0.731	0.681	94.338	0.737	0.686	71.429	0.643	0.605	70.779							
4	0.919	0.916	96.795	0.718	0.663	93.864	0.79	0.767	79.605	0.739	0.716	79.605							
segment0										vehicle0									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.975	0.975	98.918	0.991	0.991	99.567	0.917	0.911	94.706	0.959	0.924	95.294	0.87	0.865	91.765				
1	0.992	0.994	98.918	0.992	0.992	99.784	0.936	0.911	94.706	0.906	0.904	93.529							
2	0.999	0.999	99.784	0.999	0.999	99.784	0.948	0.944	94.118	0.943	0.943	95.294							
3	0.984	0.982	99.134	0.992	0.992	99.784	0.955	0.938	95.882	0.963	0.963	97.059							
4	0.966	0.968	98.913	0.977	0.977	99.348	0.932	0.934	93.976	0.928	0.926	95.783							
vehicle1										vehicle2									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.804	0.664	78.235	0.511	0.151	74.706	0.959	0.922	95.294	0.87	0.865	91.765							
1	0.666	0.586	75.882	0.557	0.337	77.059	0.98	0.98	97.059	0.939	0.938	95.294							
2	0.732	0.63	71.765	0.534	0.261	75.882	0.956	0.942	94.706	0.893	0.89	92.941							
3	0.736	0.693	72.353	0.523	0.213	75.294	0.96	0.957	95.882	0.832	0.825	88.235							
4	0.656	0.618	71.084	0.529	0.268	75.904	0.979	0.956	96.988	0.916	0.915	94.578							
vehicle3										wisconsin									
Fold	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	
0	0.706	0.626	72.353	0.5	0	74.706	0.948	0.93	93.431	0.968	0.968	97.08							
1	0.793	0.761	81.176	0.5	0	74.706	0.967	0.952	96.35	0.968	0.968	97.08							
2	0.788	0.616	78.235	0.5	0	74.706	0.877	0.92	93.431	0.958	0.957	97.08							
3	0.615	0.577	72.941	0.5	0	74.706	0.986	0.957	96.35	0.984	0.984								

ecoli-0-vs_1																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	1	0.947	93.333	0.969	0.968	97.778	IPADE-ID	0.875	0.866	91.111	0.534	0.263	40		
1	BC	1	0.965	95.556	1	1	100	IPADE-ID	1	0.851	82.222	0.672	0.587	57.778		
2	BC	0.969	0.968	97.778	0.969	0.968	97.778	IPADE-ID	0.966	0.965	95.556	0.81	0.788	75.556		
3	BC	0.966	0.965	95.556	0.983	0.983	97.778	IPADE-ID	1	1	100	0.517	0.186	37.778		
4	BC	0.962	0.961	97.5	0.962	0.961	97.5	IPADE-ID	1	0.903	87.5	0.5	0	32.5		
0	CNN	1	1	100	0.966	0.965	95.556	NCL	1	1	100	0.969	0.968	97.778		
1	CNN	0.969	0.968	97.778	0.938	0.935	95.556	NCL	1	1	100	0.938	0.935	95.556		
2	CNN	1	1	100	0.969	0.968	97.778	NCL	0.934	0.934	93.333	0.969	0.968	97.778		
3	CNN	0.969	0.968	97.778	0.969	0.968	97.778	NCL	0.969	0.968	97.778	0.969	0.968	97.778		
4	CNN	0.963	0.962	95	0.981	0.981	97.5	NCL	1	1	100	1	1	100		
0	CPM	0.969	0.968	97.778	0.938	0.935	95.556	NM	0.952	0.951	95.556	0.969	0.968	97.778		
1	CPM	1	1	100	0.969	0.968	97.778	NM	1	1	100	1	1	100		
2	CPM	1	1	100	1	1	100	NM	1	1	100	0.969	0.968	97.778		
3	CPM	0.983	0.983	97.778	0.938	0.935	95.556	NM	0.969	0.968	97.778	0.969	0.968	97.778		
4	CPM	0.962	0.961	97.5	0.962	0.961	97.5	NM	0.981	0.981	97.5	0.885	0.877	92.5		
0	ClusterOSS	0.969	0.968	97.778	0.938	0.935	95.556	OSS	0.952	0.951	95.556	0.952	0.951	95.556		
1	ClusterOSS	0.897	0.891	86.667	0.983	0.983	97.778	OSS	1	1	100	1	1	100		
2	ClusterOSS	0.828	0.809	77.778	0.897	0.891	86.667	OSS	1	1	100	0.906	0.901	93.333		
3	ClusterOSS	0.983	0.983	97.778	0.983	0.983	97.778	OSS	0.969	0.968	97.778	0.952	0.951	95.556		
4	ClusterOSS	0.943	0.943	95	0.962	0.961	97.5	OSS	1	0.981	97.5	1	1	100		
0	EE	0.983	0.983	97.778	1	1	100	RU	0.969	0.968	97.778	0.969	0.968	97.778		
1	EE	0.969	0.968	97.778	0.938	0.935	95.556	RU	1	1	100	0.969	0.968	97.778		
2	EE	0.983	0.983	97.778	0.983	0.983	97.778	RU	0.931	0.928	91.111	0.858	0.851	88.889		
3	EE	0.983	0.983	97.778	0.889	0.886	91.111	RU	0.931	0.928	91.111	0.983	0.983	97.778		
4	EE	0.962	0.961	97.5	0.925	0.925	92.5	RU	0.962	0.961	97.5	0.962	0.961	97.5		
0	ENN	0.983	0.983	97.778	1	1	100	SCB	1	1	100	0.969	0.968	97.778		
1	ENN	0.969	0.968	97.778	0.969	0.968	97.778	SCB	0.969	0.968	97.778	0.938	0.935	95.556		
2	ENN	1	1	100	0.875	0.866	91.111	SCB	0.969	0.968	97.778	0.969	0.968	97.778		
3	ENN	1	1	100	1	1	100	SCB	0.948	0.947	93.333	1	1	100		
4	ENN	0.962	0.961	97.5	0.923	0.92	95	SCB	0.963	0.943	92.5	0.962	0.961	97.5		
0	EUS	1	1	100	0.969	0.968	97.778	TL	0.969	0.968	97.778	0.938	0.935	95.556		
1	EUS	1	1	100	1	1	100	TL	0.969	0.968	97.778	0.969	0.968	97.778		
2	EUS	0.931	0.928	91.111	0.952	0.951	95.556	TL	1	1	100	1	1	100		
3	EUS	0.952	0.951	95.556	0.952	0.951	95.556	TL	1	1	100	0.906	0.901	93.333		
4	EUS	0.962	0.961	97.5	0.962	0.961	97.5	TL	0.981	0.981	97.5	0.962	0.961	97.5		
0	IHTS	1	0.965	95.556	1	1	100									
1	IHTS	0.969	0.968	97.778	0.983	0.983	97.778									
2	IHTS	0.966	0.965	95.556	1	1	100									
3	IHTS	1	1	100	1	1	100									
4	IHTS	0.925	0.925	92.5	0.887	0.887	87.5									
ecoli1																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.885	0.844	77.941	0.882	0.88	85.294	IPADE-ID	0.662	0.771	82.353	0.5	0	76.471		
1	BC	0.867	0.889	89.706	0.901	0.901	88.235	IPADE-ID	0.874	0.87	86.765	0.5	0	76.471		
2	BC	0.843	0.848	86.765	0.873	0.87	83.824	IPADE-ID	0.94	0.94	94.118	0.5	0	76.471		
3	BC	0.96	0.92	88.235	0.901	0.901	88.235	IPADE-ID	0.5	0	76.471	0.5	0	76.471		
4	BC	0.829	0.845	84.375	0.806	0.805	82.813	IPADE-ID	0.925	0.932	93.75	0.5	0	79.688		
0	CNN	0.952	0.775	88.235	0.75	0.707	88.235	NCL	0.944	0.767	86.765	0.901	0.901	88.235		
1	CNN	0.894	0.821	91.176	0.798	0.797	82.353	NCL	0.973	0.858	92.647	0.827	0.823	86.765		
2	CNN	0.978	0.791	91.176	0.875	0.866	94.118	NCL	0.924	0.857	88.235	0.901	0.901	88.235		
3	CNN	0.801	0.839	85.294	0.817	0.815	85.294	NCL	0.807	0.78	83.824	0.714	0.698	79.412		
4	CNN	0.896	0.824	92.188	0.884	0.883	90.625	NCL	0.785	0.816	90.625	0.778	0.761	87.5		
0	CPM	0.799	0.857	88.235	0.531	0.25	77.941	NM	0.918	0.767	86.765	0.786	0.78	83.824		
1	CPM	0.842	0.693	85.294	0.5	0	76.471	NM	0.969	0.93	89.706	0.95	0.949	95.588		
2	CPM	0.898	0.788	85.294	0.793	0.775	88.235	NM	0.939	0.921	91.176	0.921	0.921	91.176		
3	CPM	0.737	0.815	85.294	0.563	0.354	79.412	NM	0.934	0.93	92.647	0.846	0.841	89.706		
4	CPM	0.882	0.673	87.5	0.759	0.727	89.063	NM	0.826	0.854	85.938	0.758	0.755	79.688		
0	ClusterOSS	0.504	0.325	67.647	0.5	0	76.471	OSS	0.698	0.783	89.706	0.803	0.783	89.706		
1	ClusterOSS	0.863	0.86	82.353	0.822	0.82	79.412	OSS	0.82	0.841	89.706	0.776	0.771	82.353		
2	ClusterOSS	0.673	0.772	69.118	0.921	0.921	91.176	OSS	0.946	0.813	89.706	0.846	0.841	89.706		
3	ClusterOSS	0.671	0.67	66.176	0.531	0.25	77.941	OSS	0.947	0.783	89.706	0.834	0.821	91.176		
4	ClusterOSS	0.792	0.747	73.438	0.827	0.816	90.625	OSS	0.892	0.912	90.625	0.817	0.807	89.063		
0	EE	0.901	0.901	88.235	0.933	0.93	89.706	RU	0.913	0.877	82.353	0.933	0.93	89.706		
1	EE	0.888	0.857	88.235	0.873	0.87	83.824	RU	0.941	0.93	92.647	0.839	0.839	85.294		
2	EE	0.918	0.87	83.824	0.829	0.829	83.824	RU	0.957	0.889	89.706	0.93	0.93	92.647		
3	EE	0.944	0.911	89.706	0.839	0.839	85.294	RU	0.832	0.831	80.882	0.813	0.81	77.941		
4	EE	0.931	0.929	89.063	0.922	0.918	87.5	RU	0.903	0.882	85.938	0.912	0.907	85.938		
0	ENN	0.971	0.858	92.647	0.899	0.899	91.176	SCB	0.821	0.839	85.294	0.882	0.88	85.294		
1	ENN	0.937	0.911	89.706	0.858	0.857	88.235	SCB	0.904	0.909	86.765	0.901	0.901	88.235		
2	ENN	0.762	0.841	89.706	0.827	0.823	86.765	SCB	0.907	0.88	85.294	0.841	0.841	82.353		
3	ENN	0.928	0.875	91.176	0.767	0.763	80.882	SCB	0.911	0.911	89.706	0.849	0.848	86.765		
4	ENN	0.931	0.918	87.5	0.874	0.874	89.063	SCB	0.991	0.911	95.313	0.932	0.932	93.75		
0	EUS	0.889	0.889	89.706	0.889	0.889	89.706	TL	0.828	0.721	83.824					

ecoli2																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.895	0.889	94.118	0.947	0.946	91.176	IPADE-ID	0.748	0.767	73.529	0.5	0	83.824		
1	BC	0.781	0.758	77.941	0.785	0.783	82.353	IPADE-ID	0.736	0.692	58.824	0.5	0	83.824		
2	BC	0.93	0.927	88.235	0.904	0.898	83.824	IPADE-ID	0.5	0	83.824	0.5	0	83.824		
3	BC	0.776	0.774	80.882	0.876	0.875	85.294	IPADE-ID	0.868	0.858	77.941	0.919	0.919	92.647		
4	BC	0.982	0.982	96.875	0.946	0.945	90.625	IPADE-ID	0.964	0.964	93.75	0.5	0	87.5		
0	CNN	0.903	0.762	86.765	0.682	0.603	89.706	NCL	0.815	0.889	94.118	0.801	0.784	91.176		
1	CNN	0.634	0.603	89.706	0.5	0	83.824	NCL	0.752	0.791	92.647	0.682	0.603	89.706		
2	CNN	0.854	0.897	95.588	0.719	0.668	89.706	NCL	0.895	0.889	94.118	0.727	0.674	91.176		
3	CNN	0.76	0.692	82.353	0.682	0.603	89.706	NCL	0.904	0.889	94.118	0.865	0.864	89.706		
4	CNN	0.636	0.783	93.75	0.875	0.866	96.875	NCL	0.873	0.927	96.875	0.804	0.783	93.75		
0	CPM	0.687	0.634	70.588	0.739	0.732	80.882	NM	0.904	0.889	82.353	0.816	0.795	69.118		
1	CPM	0.765	0.776	89.706	0.5	0	83.824	NM	0.937	0.875	85.294	0.876	0.875	85.294		
2	CPM	0.575	0.705	85.294	0.545	0.302	85.294	NM	0.591	0.644	83.824	0.839	0.839	85.294		
3	CPM	0.871	0.911	91.176	0.5	0	83.824	NM	0.785	0.667	58.824	0.886	0.879	80.882		
4	CPM	0.876	0.791	95.313	0.866	0.858	95.313	NM	0.808	0.76	65.75	0.759	0.75	67.188		
0	ClusterOSS	0.832	0.828	77.941	0.876	0.875	85.294	OSS	0.869	0.838	92.647	0.773	0.739	92.647		
1	ClusterOSS	0.921	0.918	86.765	0.921	0.918	86.765	OSS	0.961	0.853	95.588	0.682	0.603	89.706		
2	ClusterOSS	0.5	0	83.824	0.5	0	83.824	OSS	0.951	0.945	97.059	0.855	0.845	94.118		
3	ClusterOSS	0.765	0.784	91.176	0.5	0	83.824	OSS	0.704	0.762	86.765	0.755	0.725	89.706		
4	ClusterOSS	0.796	0.579	82.813	0.911	0.91	93.75	OSS	0.721	0.791	95.313	0.75	0.707	93.75		
0	EE	0.878	0.838	92.647	0.919	0.919	92.647	RU	0.726	0.767	73.529	0.884	0.884	86.765		
1	EE	0.949	0.908	85.294	0.956	0.955	92.647	RU	0.911	0.866	83.824	0.921	0.918	86.765		
2	EE	0.781	0.766	79.412	0.911	0.911	91.176	RU	0.837	0.818	76.471	0.858	0.857	82.353		
3	EE	0.886	0.806	70.588	0.886	0.879	80.882	RU	0.88	0.872	91.176	0.965	0.964	94.118		
4	EE	0.859	0.857	84.375	0.857	0.857	84.375	RU	0.879	0.901	92.188	0.848	0.848	82.813		
0	ENN	0.907	0.897	95.588	0.766	0.755	85.294	SCB	0.895	0.889	94.118	0.764	0.732	91.176		
1	ENN	0.877	0.872	91.176	0.656	0.587	85.294	SCB	0.821	0.791	92.647	0.71	0.662	88.235		
2	ENN	0.629	0.668	89.706	0.656	0.587	85.294	SCB	0.884	0.732	91.176	0.846	0.838	92.647		
3	ENN	1	100	0.955	0.953	98.529	SCB	0.859	0.798	94.118	0.755	0.725	89.706			
4	ENN	0.991	0.991	98.438	0.866	0.858	95.313	SCB	0.927	0.919	95.313	0.92	0.919	95.313		
0	EUS	0.788	0.804	79.412	0.93	0.927	88.235	TL	0.794	0.587	85.294	0.764	0.732	91.176		
1	EUS	0.803	0.799	85.294	0.874	0.872	91.176	TL	0.867	0.897	95.588	0.755	0.725	89.706		
2	EUS	0.9	0.864	89.706	0.893	0.893	88.235	TL	0.618	0.668	89.706	0.719	0.668	89.706		
3	EUS	0.892	0.893	88.235	0.947	0.946	91.176	TL	0.878	0.905	97.059	0.818	0.798	94.118		
4	EUS	0.874	0.857	84.375	0.938	0.935	89.063	TL	0.763	0.866	96.875	0.875	0.866	96.875		
0	IHTS	0.585	0.536	42.647	0.5	0	16.176									
1	IHTS	0.5	0	16.176	0.509	0.132	17.647									
2	IHTS	0.711	0.649	51.471	0.5	0	16.176									
3	IHTS	0.509	0.132	17.647	0.509	0.132	17.647									
4	IHTS	0.864	0.567	40.625	0.723	0.668	51.563									
ecoli3																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.783	0.78	83.824	0.814	0.813	77.941	IPADE-ID	0.902	0.896	82.353	0.5	0	89.706		
1	BC	0.722	0.701	69.118	0.806	0.804	76.471	IPADE-ID	0.957	0.903	94.118	0.926	0.923	86.765		
2	BC	0.967	0.923	86.765	0.861	0.849	75	IPADE-ID	0.939	0.75	94.118	0.737	0.718	86.765		
3	BC	0.755	0.742	76.471	0.959	0.958	92.647	IPADE-ID	0.836	0.82	70.588	0.639	0.528	35.294		
4	BC	0.614	0.607	81.25	0.912	0.908	84.375	IPADE-ID	0.906	0.918	96.875	0.772	0.737	59.375		
0	CNN	0.715	0.649	92.647	0.5	0	89.706	NCL	0.915	0.756	95.588	0.5	0	89.706		
1	CNN	0.642	0.521	88.235	0.5	0	89.706	NCL	0.937	0.824	92.647	0.571	0.378	91.176		
2	CNN	0.642	0.649	92.647	0.5	0	89.706	NCL	0.66	0.638	89.706	0.5	0	89.706		
3	CNN	0.5	0	89.706	0.5	0	89.706	NCL	0.66	0.535	92.647	0.5	0	89.706		
4	CNN	0.768	0.743	92.188	0.5	0	89.063	NCL	0.841	0.823	92.188	0.5	0	89.063		
0	CPM	0.681	0.633	88.235	0.555	0.372	88.235	NM	0.804	0.335	20.588	0.666	0.638	51.471		
1	CPM	0.5	0	10.294	0.665	0.622	85.294	NM	0.629	0.644	91.176	0.726	0.726	73.529		
2	CPM	0.514	0.433	61.765	0.5	0	89.706	NM	0.615	0.633	88.235	0.644	0.64	58.824		
3	CPM	0.828	0.81	69.118	0.5	0	89.706	NM	0.741	0.718	86.765	0.523	0.521	48.529		
4	CPM	0.759	0.736	90.625	0.563	0.375	89.063	NM	0.742	0.722	87.5	0.655	0.653	60.938		
0	ClusterOSS	0.773	0.768	70.588	0.806	0.804	76.471	OSS	0.81	0.535	92.647	0.5	0	89.706		
1	ClusterOSS	0.882	0.701	54.412	0.885	0.878	79.412	OSS	0.951	0.895	92.647	0.5	0	89.706		
2	ClusterOSS	0.889	0.838	82.353	0.893	0.887	80.882	OSS	0.745	0.627	86.765	0.547	0.369	86.765		
3	ClusterOSS	0.704	0.691	80.882	0.5	0	89.706	OSS	0.694	0.638	89.706	0.5	0	89.706		
4	ClusterOSS	0.787	0.784	84.375	0.823	0.823	79.688	OSS	0.801	0.838	95.313	0.5	0	89.063		
0	EE	0.844	0.83	80.882	0.893	0.887	80.882	RU	0.861	0.849	75	0.861	0.849	75		
1	EE	0.902	0.896	82.353	0.918	0.914	85.294	RU	0.85	0.757	79.412	0.877	0.868	77.941		
2	EE	0.943	0.887	91.176	0.838	0.838	82.353	RU	0.959	0.941	89.706	0.838	0.838	82.353		
3	EE	0.926	0.923	86.765	0.926	0.923	86.765	RU	0.837	0.804	76.471	0.847	0.847	83.824		
4	EE	0.762	0.756	68.75	0.771	0.766	70.313	RU	0.608	0.701	82.813	0.895	0.889	81.25		
0	ENN	0.986	0.918	97.059	0.761	0.737	91.176	SCB	0.77	0.622	85.294	0.896	0.895	92.647		
1	ENN	0.753	0.743	92.647	0.706	0.649	92.647	SCB	0.938	0.81	89.706	0.753	0.731	89.706		
2	ENN	0.905	0.838	95.588	0.643	0.535	92.647	SCB	0.858	0.817	91.176	0.824	0.817	91.176		
3	ENN	0.674	0.526	89.706	0.563	0.375	89.706	SCB	0.92	0.817	91.176	0.841	0.831	94.118		
4	ENN	0.798	0.799	87.5	0.634	0.53	90.625	SCB	0.823	0.823	92.188	0.777	0.749	93.75	</	

glasso-0-1-2-3_vs_4-5-6																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.864	0.862	88.636	0.939	0.939	95.455	IPADE-ID	0.788	0.773	86.364	0.5	0	25		
1	BC	0.879	0.877	90.909	0.833	0.827	88.636	IPADE-ID	0.814	0.818	81.818	0.5	0	75		
2	BC	0.909	0.905	95.455	0.939	0.939	95.455	IPADE-ID	0.803	0.799	84.091	0.5	0	25		
3	BC	0.979	0.937	90.909	0.848	0.848	86.364	IPADE-ID	0.758	0	75	0.5	0	25		
4	BC	0.896	0.88	89.474	0.864	0.864	86.842	IPADE-ID	0.922	0.756	92.105	0.5	0	81.579		
0	CNN	0.669	0.573	54.545	0.758	0.727	86.364	NCL	0.927	0.924	93.182	0.909	0.905	95.455		
1	CNN	0.879	0.87	81.818	0.697	0.653	81.818	NCL	0.97	0.969	95.455	0.803	0.799	84.091		
2	CNN	0.742	0.786	81.818	0.848	0.84	90.909	NCL	0.848	0.84	90.909	0.879	0.877	90.909		
3	CNN	0.843	0.799	84.091	0.818	0.798	90.909	NCL	0.904	0.905	95.455	0.909	0.905	95.455		
4	CNN	0.919	0.911	94.737	0.912	0.911	94.737	NCL	0.707	0.831	92.105	0.929	0.926	97.368		
0	CPM	0.864	0.862	84.091	0.924	0.924	93.182	NM	0.966	0.924	93.182	0.939	0.939	95.455		
1	CPM	0.727	0.722	68.182	0.667	0.577	50	NM	0.883	0.924	93.182	0.818	0.813	86.364		
2	CPM	0.803	0.796	75	0.758	0.727	86.364	NM	0.891	0.877	90.909	0.879	0.877	90.909		
3	CPM	0.848	0.848	86.364	0.758	0.757	77.273	NM	0.886	0.878	86.364	0.894	0.894	88.636		
4	CPM	0.88	0.789	84.211	0.896	0.895	92.105	NM	0.908	0.911	94.737	0.896	0.895	92.105		
0	ClusterOSS	0.788	0.786	81.818	0.712	0.664	84.091	OSS	0.913	0.953	97.727	0.909	0.905	95.455		
1	ClusterOSS	0.742	0.735	79.545	0.909	0.905	95.455	OSS	0.69	0.761	84.091	0.848	0.835	77.273		
2	ClusterOSS	0.879	0.877	90.909	0.894	0.891	93.182	OSS	0.905	0.833	84.091	0.909	0.909	90.909		
3	ClusterOSS	0.924	0.921	88.636	0.894	0.894	88.636	OSS	0.955	0.953	97.727	0.864	0.853	93.182		
4	ClusterOSS	0.954	0.817	89.474	0.786	0.756	92.105	OSS	0.903	0.831	92.105	0.793	0.789	84.211		
0	EE	0.956	0.921	88.636	0.924	0.924	93.182	RU	0.939	0.937	90.909	0.848	0.848	86.364		
1	EE	0.939	0.937	90.909	0.864	0.853	93.182	RU	0.97	0.969	95.455	0.924	0.924	93.182		
2	EE	0.97	0.969	95.455	0.939	0.939	95.455	RU	0.904	0.798	90.909	0.864	0.853	93.182		
3	EE	0.935	0.891	93.182	0.879	0.877	90.909	RU	0.93	0.937	90.909	0.879	0.877	90.909		
4	EE	0.793	0.789	84.211	0.793	0.789	84.211	RU	0.952	0.95	92.105	0.88	0.88	89.474		
0	ENN	0.955	0.939	95.455	0.773	0.761	84.091	SCB	1	0.953	97.727	0.924	0.924	93.182		
1	ENN	0.97	0.969	95.455	0.924	0.924	93.182	SCB	0.824	0.848	86.364	0.848	0.84	90.909		
2	ENN	0.809	0.786	88.636	0.788	0.773	86.364	SCB	0.985	0.985	97.727	0.864	0.853	93.182		
3	ENN	0.985	0.985	97.727	0.955	0.953	93.182	SCB	0.853	0.84	90.909	0.788	0.773	86.364		
4	ENN	0.984	0.984	97.368	0.786	0.756	92.105	SCB	0.968	0.967	94.737	0.912	0.911	94.737		
0	EUS	0.879	0.877	90.909	0.939	0.939	95.455	TL	0.875	0.924	93.182	0.985	0.985	97.727		
1	EUS	0.879	0.877	90.909	0.894	0.891	93.182	TL	0.953	0.953	97.727	0.864	0.853	93.182		
2	EUS	0.829	0.848	86.364	0.864	0.862	88.636	TL	0.806	0.84	90.909	0.758	0.727	86.364		
3	EUS	0.909	0.888	84.091	0.864	0.862	88.636	TL	0.982	0.891	93.182	0.909	0.905	95.455		
4	EUS	0.892	0.864	86.842	0.753	0.731	86.842	TL	0.977	0.967	94.737	0.952	0.95	92.105		
0	IHTS	0.848	0.846	81.818	0.924	0.924	93.182									
1	IHTS	0.894	0.921	88.636	0.864	0.853	93.182									
2	IHTS	0.855	0.891	93.182	0.879	0.877	90.909									
3	IHTS	0.822	0.853	93.182	0.848	0.84	90.909									
4	IHTS	0.468	0.359	28.947	0.5	0	18.421									
glasso																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.783	0.78	83.824	0.814	0.813	77.941	IPADE-ID	0.5	0	67.442	0.5	0	67.442		
1	BC	0.722	0.701	69.118	0.806	0.804	76.471	IPADE-ID	0.638	0	67.442	0.5	0	67.442		
2	BC	0.967	0.923	86.765	0.861	0.849	75	IPADE-ID	0.707	0.643	60.465	0.5	0	32.558		
3	BC	0.755	0.742	76.471	0.958	0.958	92.647	IPADE-ID	0.672	0.587	55.814	0.5	0	67.442		
4	BC	0.614	0.607	81.25	0.912	0.908	84.375	IPADE-ID	0.607	0	66.667	0.5	0	33.333		
0	CNN	0.715	0.649	92.647	0.5	0	89.706	NCL	0.855	0.842	83.721	0.776	0.743	69.767		
1	CNN	0.642	0.521	88.235	0.5	0	89.706	NCL	0.781	0.555	69.767	0.647	0.63	69.767		
2	CNN	0.642	0.649	92.647	0.5	0	89.706	NCL	0.631	0.657	74.419	0.611	0.583	67.442		
3	CNN	0.5	0	89.706	0.5	0	89.706	NCL	0.888	0.86	86.047	0.592	0.544	67.442		
4	CNN	0.768	0.743	92.188	0.5	0	89.063	NCL	0.719	0.699	69.048	0.607	0.515	71.429		
0	CPM	0.681	0.633	88.235	0.555	0.372	88.235	NM	0.647	0.628	60.465	0.49	0.486	51.163		
1	CPM	0.5	0	10.294	0.665	0.622	85.294	NM	0.772	0.719	72.093	0.494	0.471	44.186		
2	CPM	0.514	0.433	61.765	0.5	0	89.706	NM	0.756	0.709	67.442	0.486	0.403	58.14		
3	CPM	0.828	0.81	69.118	0.5	0	89.706	NM	0.729	0.658	69.767	0.453	0.421	51.163		
4	CPM	0.759	0.736	90.625	0.563	0.375	89.063	NM	0.842	0.794	76.19	0.429	0.319	52.381		
0	ClusterOSS	0.773	0.768	70.588	0.806	0.804	76.471	OSS	0.619	0.63	69.767	0.607	0.463	74.419		
1	ClusterOSS	0.882	0.701	54.412	0.885	0.878	79.412	OSS	0.813	0.855	88.372	0.536	0.267	69.767		
2	ClusterOSS	0.889	0.838	82.353	0.893	0.887	80.882	OSS	0.881	0.858	83.721	0.5	0	67.442		
3	ClusterOSS	0.704	0.691	80.882	0.5	0	89.706	OSS	0.867	0.858	83.721	0.59	0.455	72.093		
4	ClusterOSS	0.787	0.784	84.375	0.823	0.823	79.688	OSS	0.714	0.73	71.429	0.607	0.553	69.048		
0	EE	0.844	0.84	80.882	0.893	0.887	80.882	RU	0.924	0.82	79.07	0.707	0.643	60.465		
1	EE	0.902	0.896	82.353	0.918	0.914	85.294	RU	0.867	0.769	74.419	0.672	0.587	55.814		
2	EE	0.943	0.887	91.176	0.838	0.838	82.353	RU	0.807	0.789	79.07	0.724	0.67	62.791		
3	EE	0.926	0.923	86.765	0.926	0.923	86.765	RU	0.789	0.772	76.744	0.707	0.643	60.465		
4	EE	0.762	0.756	68.75	0.771	0.766	70.313	RU	0.749	0.763	73.81	0.75	0.707	66.667		
0	ENN	0.986	0.918	97.059	0.761	0.737	91.176	SCB	0.855	0.825	81.395	0.724	0.67	62.791		
1	ENN	0.753	0.743	92.647	0.706	0.649	92.647	SCB	0.7	0.729	69.767	0.663	0.62	74.419		
2	ENN	0.905	0.838	95.588	0.643	0.535	92.647	SCB	0.674	0.702	76.744	0.69	0.616	58.14		
3	ENN	0.674	0.526	89.706	0.563	0										

glass1																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.702	0.743	72.727	0.518	0.463	45.455	IPADE-ID	0.5	0	36.364	0.5	0	63.636		
1	BC	0.756	0.685	65.909	0.625	0.5	52.273	IPADE-ID	0.6	0.245	63.636	0.5	0	63.636		
2	BC	0.701	0.685	70.455	0.647	0.579	56.818	IPADE-ID	0.567	0.511	50	0.5	0	36.364		
3	BC	0.709	0.627	68.182	0.612	0.518	52.273	IPADE-ID	0.558	0.409	45.455	0.5	0	36.364		
4	BC	0.583	0.519	65.789	0.609	0.566	52.632	IPADE-ID	0.5	0	68.421	0.5	0	68.421		
0	CNN	0.5	0	63.636	0.563	0.354	68.182	NCL	0.757	0.718	72.727	0.531	0.25	65.909		
1	CNN	0.5	0	36.364	0.665	0.607	59.091	NCL	0.597	0.579	70.455	0.545	0.347	65.909		
2	CNN	0.5	0	63.636	0.504	0.392	59.091	NCL	0.705	0.717	75	0.446	0	56.818		
3	CNN	0.5	0	63.636	0.513	0.245	63.636	NCL	0.674	0.618	63.636	0.5	0	63.636		
4	CNN	0.5	0	68.421	0.404	0	55.263	NCL	0.721	0.734	76.316	0.5	0	68.421		
0	CPM	0.531	0.25	65.909	0.5	0	63.636	NM	0.594	0.616	61.364	0.509	0.491	54.545		
1	CPM	0.586	0.573	63.636	0.5	0	63.636	NM	0.628	0.66	63.636	0.549	0.549	54.545		
2	CPM	0.598	0.53	52.273	0.549	0.482	47.727	NM	0.709	0.586	56.818	0.33	0.259	38.636		
3	CPM	0.647	0.646	63.636	0.509	0.496	47.727	NM	0.568	0.565	54.545	0.446	0.401	50		
4	CPM	0.5	0	31.579	0.462	0	63.158	NM	0.889	0.84	84.211	0.558	0.555	57.895		
0	ClusterOSS	0.682	0.637	61.364	0.5	0	36.364	OSS	0.731	0.723	79.545	0.5	0	63.636		
1	ClusterOSS	0.527	0.395	43.182	0.585	0.539	52.273	OSS	0.853	0.852	86.364	0.5	0	63.636		
2	ClusterOSS	0.556	0.5	52.273	0.478	0.236	59.091	OSS	0.645	0.601	61.364	0.5	0	63.636		
3	ClusterOSS	0.536	0.616	61.364	0.518	0.463	45.455	OSS	0.8	0.768	77.273	0.701	0.701	70.455		
4	ClusterOSS	0.482	0.474	50	0.474	0.31	34.211	OSS	0.756	0.721	71.053	0.5	0	68.421		
0	EE	0.574	0.584	59.091	0.464	0.366	38.636	RU	0.831	0.762	75	0.571	0.543	52.273		
1	EE	0.743	0.709	77.273	0.598	0.53	52.273	RU	0.584	0.675	65.909	0.607	0.463	50		
2	EE	0.824	0.768	77.273	0.612	0.518	52.273	RU	0.749	0.729	70.455	0.625	0.5	52.273		
3	EE	0.698	0.616	61.364	0.612	0.518	52.273	RU	0.738	0.735	75	0.5	0.433	43.182		
4	EE	0.795	0.832	78.947	0.615	0.48	47.368	RU	0.696	0.7	68.421	0.612	0.531	50		
0	ENN	0.695	0.701	70.455	0.478	0.236	59.091	SCB	0.698	0.717	75	0.545	0.539	52.273		
1	ENN	0.622	0.65	77.273	0.482	0	61.364	SCB	0.64	0.612	65.909	0.5	0	63.636		
2	ENN	0.681	0.425	68.182	0.5	0	63.636	SCB	0.746	0.75	72.727	0.567	0.507	63.636		
3	ENN	0.778	0.785	79.545	0.513	0.245	63.636	SCB	0.824	0.791	77.273	0.545	0.347	65.909		
4	ENN	0.683	0.66	65.789	0.462	0	63.158	SCB	0.705	0.698	71.053	0.545	0.531	50		
0	EUS	0.664	0.646	63.636	0.625	0.5	52.273	TL	0.618	0.655	72.727	0.5	0	63.636		
1	EUS	0.834	0.781	77.273	0.629	0.549	54.545	TL	0.779	0.751	77.273	0.5	0	63.636		
2	EUS	0.674	0.701	70.455	0.603	0.565	54.545	TL	0.696	0.681	77.273	0.5	0	63.636		
3	EUS	0.711	0.665	65.909	0.527	0.395	43.182	TL	0.788	0.762	75	0.5	0	63.636		
4	EUS	0.78	0.76	76.316	0.631	0.563	52.632	TL	0.684	0.734	76.316	0.5	0	68.421		
0	IHTS	0.701	0.267	40.909	0.5	0	36.364									
1	IHTS	0.52	0.586	56.818	0.554	0.327	43.182									
2	IHTS	0.5	0	36.364	0.5	0	36.364									
3	IHTS	0.67	0.586	56.818	0.5	0	36.364									
4	IHTS	0.723	0.594	55.263	0.5	0	31.579									
glass6																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.849	0.849	86.047	0.863	0.862	88.372	IPADE-ID	0.903	0	13.953	0.5	0	13.953		
1	BC	0.822	0.822	81.395	0.736	0.697	90.696	IPADE-ID	0.919	0.435	30.233	0.5	0	86.047		
2	BC	0.876	0.875	90.698	0.89	0.888	93.023	IPADE-ID	0.917	0	13.953	0.5	0	13.953		
3	BC	0.878	0.87	79.07	0.919	0.919	86.047	IPADE-ID	0.928	0.545	39.355	0.903	0.9	95.349		
4	BC	0.946	0.93	88.095	0.959	0.959	92.857	IPADE-ID	1	0.285	19.048	0.486	0	85.714		
0	CNN	0.824	0.805	69.767	0.89	0.888	93.023	NCL	0.806	0.794	90.698	0.806	0.794	90.698		
1	CNN	0.869	0.765	72.093	0.486	0	83.721	NCL	0.903	0.9	95.349	0.833	0.816	95.349		
2	CNN	0.703	0.637	48.837	0.5	0	86.047	NCL	0.903	0.9	95.349	0.903	0.9	95.349		
3	CNN	0.77	0.735	60.465	0.903	0.9	95.349	NCL	0.822	0.9	95.349	0.917	0.913	97.674		
4	CNN	0.895	0.593	42.857	0.759	0.743	88.095	NCL	0.986	0.959	92.857	0.973	0.973	95.238		
0	CPM	0.66	0.403	27.907	0.5	0	86.047	NM	0.82	0.805	93.023	0.82	0.805	93.023		
1	CPM	0.718	0.72	65.116	0.653	0.569	88.372	NM	0.917	0.913	97.674	0.917	0.913	97.674		
2	CPM	0.878	0.87	79.07	0.583	0.408	88.372	NM	0.851	0.838	74.419	0.973	0.973	95.349		
3	CPM	0.809	0.836	83.721	0.5	0	86.047	NM	0.838	0.822	72.093	0.986	0.986	97.674		
4	CPM	0.8	0.775	95.238	0.5	0	88.095	NM	0.959	0.959	92.857	0.986	0.986	97.619		
0	ClusterOSS	0.959	0.915	86.047	0.986	0.986	97.674	OSS	0.876	0.875	90.698	0.82	0.805	93.023		
1	ClusterOSS	0.806	0.794	90.698	0.903	0.9	95.349	OSS	0.622	0.788	67.442	0.986	0.986	97.674		
2	ClusterOSS	0.836	0.836	83.721	0.863	0.862	88.372	OSS	1	1	100	0.917	0.913	97.674		
3	ClusterOSS	0.986	0.986	97.674	0.986	0.986	97.674	OSS	0.667	0.816	95.349	0.917	0.913	97.674		
4	ClusterOSS	0.946	0.944	90.476	0.824	0.805	69.048	OSS	0.773	0.753	90.476	0.786	0.764	92.857		
0	EE	0.903	0.8	95.349	0.917	0.913	97.674	RU	0.892	0.9	95.349	0.833	0.816	95.349		
1	EE	0.892	0.836	83.721	0.903	0.9	95.349	RU	0.782	0.78	74.419	0.917	0.913	97.674		
2	EE	0.973	0.973	95.349	0.973	0.973	95.349	RU	0.863	0.862	88.372	0.903	0.9	95.349		
3	EE	0.986	0.986	97.674	0.876	0.875	90.698	RU	0.946	0.944	90.698	0.973	0.973	95.349		
4	EE	0.868	0.792	78.571	0.873	0.87	92.857	RU	1	1	100	0.946	0.944	90.476		
0	ENN	1	1	100	1	1	100	SCB	0.946	0.959	93.023	0.986	0.986	97.674		
1	ENN	0.89	0.888	93.023	0.903	0.9	95.349	SCB	0.973	0.959	93.023	0.973	0.973	95.349		
2	ENN	0.973	0.93	88.372	0.986	0.986	97.674	SCB	0.806	0.794	90.698	0.82	0.805	93.023		
3	ENN	0.755	0.816	95.349	0.833	0.816	95.349	SCB	0.833	0.816	95.349	0.833	0.816	95.349		
4	ENN	0.881	0.857	90.476	0.859	0.857	90.476	SCB	1	0.986	97.619	0.9	0.894	97.619		
0	EUS	0.917	0.913	97.674	0.9											

haberman																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.704	0.689	70.968	0.584	0.558	66.129	IPADE-ID	0.564	0	27.419	0.5	0	27.419		
1	BC	0.603	0.614	66.129	0.62	0.614	66.129	IPADE-ID	0.705	0	27.419	0.5	0	27.419		
2	BC	0.552	0.546	70.968	0.708	0.708	70.968	IPADE-ID	0.586	0.574	53.226	0.5	0	27.419		
3	BC	0.665	0.651	72.581	0.621	0.56	74.194	IPADE-ID	0.678	0.663	61.29	0.5	0	27.419		
4	BC	0.603	0.562	72.414	0.544	0.379	75.862	IPADE-ID	0.69	0.672	60.345	0.5	0	22.414		
0	CNN	0.727	0.726	70.968	0.5	0	72.581	NCL	0.595	0.574	69.355	0.5	0	72.581		
1	CNN	0.579	0.575	54.839	0.489	0	70.968	NCL	0.637	0.626	59.677	0.478	0	69.355		
2	CNN	0.531	0.531	56.452	0.5	0	72.581	NCL	0.637	0.524	75.806	0.5	0	72.581		
3	CNN	0.5	0	72.581	0.5	0	72.581	NCL	0.507	0.596	67.742	0.5	0	72.581		
4	CNN	0.5	0	77.586	0.5	0	77.586	NCL	0.568	0.379	75.862	0.538	0.277	79.31		
0	CPM	0.555	0.406	72.581	0.478	0	69.355	NM	0.659	0.605	61.29	0.698	0.677	77.419		
1	CPM	0.5	0	72.581	0.5	0	72.581	NM	0.601	0.578	64.516	0.643	0.574	77.419		
2	CPM	0.646	0.622	72.581	0.5	0	72.581	NM	0.637	0.603	69.355	0.65	0.605	75.806		
3	CPM	0.5	0	72.581	0.5	0	72.581	NM	0.512	0.539	69.355	0.554	0.516	64.516		
4	CPM	0.5	0	77.586	0.5	0	77.586	NM	0.681	0.62	58.621	0.58	0.579	60.345		
0	ClusterOSS	0.584	0.583	53.226	0.54	0.446	67.742	OSS	0.5	0	72.581	0.5	0	72.581		
1	ClusterOSS	0.609	0.604	64.516	0.5	0	27.419	OSS	0.569	0	72.581	0.5	0	72.581		
2	ClusterOSS	0.551	0.452	69.355	0.5	0	72.581	OSS	0.578	0.59	72.581	0.5	0	72.581		
3	ClusterOSS	0.641	0.616	62.903	0.5	0	27.419	OSS	0.606	0.553	72.581	0.5	0	72.581		
4	ClusterOSS	0.509	0.506	53.448	0.5	0	22.414	OSS	0.731	0.648	81.034	0.489	0	75.862		
0	EE	0.591	0.578	64.516	0.58	0.505	70.968	RU	0.62	0.614	58.065	0.58	0.569	62.903		
1	EE	0.691	0.655	79.032	0.636	0.536	79.032	RU	0.68	0.647	77.419	0.639	0.637	66.129		
2	EE	0.531	0.515	58.065	0.555	0.406	72.581	RU	0.686	0.69	66.129	0.513	0.511	53.226		
3	EE	0.607	0.614	58.065	0.518	0.434	64.516	RU	0.503	0.457	59.677	0.522	0.391	67.742		
4	EE	0.777	0.788	75.862	0.703	0.683	79.31	RU	0.554	0.496	68.966	0.57	0.539	67.241		
0	ENN	0.624	0.485	66.129	0.537	0.335	72.581	SCB	0.697	0.699	72.581	0.566	0.411	74.194		
1	ENN	0.666	0.647	67.742	0.588	0.42	77.419	SCB	0.641	0.632	69.355	0.573	0.463	72.581		
2	ENN	0.576	0.539	69.355	0.573	0.463	72.581	SCB	0.648	0.66	74.194	0.5	0	72.581		
3	ENN	0.629	0.574	77.419	0.529	0.243	74.194	SCB	0.621	0.553	72.581	0.518	0.24	72.581		
4	ENN	0.607	0.628	68.966	0.544	0.379	75.862	SCB	0.532	0.531	65.517	0.489	0	75.862		
0	EUS	0.617	0.582	70.968	0.496	0.234	69.355	TL	0.581	0.511	72.581	0.5	0	72.581		
1	EUS	0.58	0.569	62.903	0.54	0.446	67.742	TL	0.674	0.676	72.581	0.5	0	72.581		
2	EUS	0.661	0.657	69.355	0.665	0.587	80.645	TL	0.529	0.44	66.129	0.5	0	72.581		
3	EUS	0.684	0.678	69.355	0.694	0.686	74.194	TL	0.5	0	72.581	0.5	0	72.581		
4	EUS	0.57	0.549	82.759	0.681	0.613	84.483	TL	0.632	0.624	75.862	0.5	0	77.586		
0	IHTS	0.542	0.542	54.839	0.5	0	27.419									
1	IHTS	0.508	0.343	33.871	0.5	0	27.419									
2	IHTS	0.489	0.149	29.032	0.5	0	27.419									
3	IHTS	0.482	0.145	27.419	0.5	0	27.419									
4	IHTS	0.5	0	22.414	0.5	0	22.414									
iris0																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	1	1	100	1	1	100	IPADE-ID	0.5	0	66.667	0.5	0	66.667		
1	BC	1	1	100	1	1	100	IPADE-ID	0.5	0	66.667	0.75	0.707	83.333		
2	BC	0.95	0.949	96.667	1	1	100	IPADE-ID	0.5	0	66.667	0.5	0	66.667		
3	BC	1	1	100	1	1	100	IPADE-ID	0.8	0.775	86.667	0.5	0	66.667		
4	BC	1	1	100	1	1	100	IPADE-ID	0.5	0	66.667	0.55	0.316	70		
0	CNN	0.5	0	33.333	0.95	0.949	96.667	NCL	1	1	100	1	1	100		
1	CNN	0.5	0	33.333	1	1	100	NCL	0.95	0.949	96.667	1	1	100		
2	CNN	0.5	0	33.333	1	1	100	NCL	1	1	100	1	1	100		
3	CNN	0.5	0	33.333	1	1	100	NCL	1	1	100	1	1	100		
4	CNN	0.5	0	33.333	1	1	100	NCL	1	1	100	1	1	100		
0	CPM	0.5	0	33.333	0.375	0.255	46.667	NM	1	1	100	1	1	100		
1	CPM	0.5	0	33.333	1	1	100	NM	1	1	100	1	1	100		
2	CPM	0.5	0	33.333	1	1	100	NM	0.95	0.949	96.667	1	1	100		
3	CPM	0.5	0	33.333	0.975	0.975	96.667	NM	1	1	100	1	1	100		
4	CPM	0.5	0	33.333	1	1	100	NM	1	1	100	1	1	100		
0	ClusterOSS	1	1	100	1	1	100	OSS	0.5	0	33.333	0.975	0.975	96.667		
1	ClusterOSS	0.65	0.548	76.667	1	1	100	OSS	1	1	100	1	1	100		
2	ClusterOSS	0.5	0	66.667	0.7	0.632	80	OSS	1	1	100	1	1	100		
3	ClusterOSS	0.8	0.775	86.667	1	1	100	OSS	0.5	0	33.333	1	1	100		
4	ClusterOSS	0.5	0	66.667	0.85	0.837	90	OSS	0.5	0	33.333	1	1	100		
0	EE	1	1	100	1	1	100	RU	1	1	100	1	1	100		
1	EE	1	1	100	1	1	100	RU	0.95	0.949	96.667	1	1	100		
2	EE	1	1	100	1	1	100	RU	1	1	100	1	1	100		
3	EE	1	1	100	1	1	100	RU	1	1	100	1	1	100		
4	EE	0.95	0.949	96.667	1	1	100	RU	1	1	100	1	1	100		
0	ENN	1	1	100	1	1	100	SCB	0.9	0.894	93.333	1	1	100		
1	ENN	0.9	0.894	93.333	1	1	100	SCB	1	1	100	1	1	100		
2	ENN	1	1	100	1	1	100	SCB	1	1	100	1	1	100		
3	ENN	1	1	100	1	1	100	SCB	1	1	100	1	1	100		
4	ENN	1	1	100	1	1	100	SCB	1	1	100	1	1	100		
0	EUS	1	1	100	1	1	100	TL	1	1	100	1	1	100		
1	EUS	1	1	100	1	1	100	TL	0.95	0.949	96.667	1	1	100		
2	EUS	1	1	100	1	1	100	TL	0.95	0.949	96.667	1	1	100		
3	EUS	1	1	100	1	1	100	TL	0.95	0.949	96.667	1	1	100		
4	EUS	0.9	0.894	93.333	1	1	100	TL	1	1	100	1	1	100		
0	IHTS	1	1	100	1	1	100	TL	1	1	100	1	1	100		
1	IHTS	1	1	100	1	1	100	TL	1	1	100	1	1	100		
2	IHTS	1	1	100	1	1	100	TL	1	1	100	1	1	100		
3	IHTS	0.95	0.949	96.667	1	1	100	TL	1	1	100	1	1	100		

		new-thyroid1															
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%			
0	BC	0.925	0.926	97.674	0.958	0.957	93.023	IPADE-ID	0.5	0	16.279	0.714	0.655	90.698			
1	BC	0.958	0.957	93.023	1	1	100	IPADE-ID	0.901	0.9	93.023	0.714	0.655	90.698			
2	BC	0.944	0.943	90.698	0.986	0.986	97.674	IPADE-ID	0.772	0.745	90.698	0.736	0.687	55.814			
3	BC	1	1	100	0.929	0.926	97.674	IPADE-ID	0.514	0.167	18.605	0.643	0.535	88.372			
4	BC	0.929	0.926	97.674	0.929	0.926	97.674	IPADE-ID	0.516	0.356	76.744	0.643	0.535	88.372			
0	CNN	0.901	0.9	93.023	0.972	0.972	95.349	NCL	0.877	0.873	88.372	0.571	0.378	86.047			
1	CNN	0.972	0.972	95.349	0.889	0.882	81.395	NCL	1	1	100	0.929	0.926	97.674			
2	CNN	0.873	0.873	88.372	0.931	0.928	88.372	NCL	0.98	0.986	97.674	0.786	0.756	93.023			
3	CNN	0.972	0.972	95.349	0.917	0.913	86.047	NCL	0.966	0.972	95.349	0.786	0.756	93.023			
4	CNN	0.774	0.772	81.395	0.714	0.655	90.698	NCL	0.927	0.913	95.349	0.857	0.845	95.349			
0	CPM	1	1	100	1	1	100	NM	0.976	0.913	95.349	0.929	0.926	97.674			
1	CPM	0.829	0.821	90.698	0.875	0.866	79.07	NM	0.929	0.926	97.674	1	1	100			
2	CPM	0.859	0.859	86.047	0.639	0.527	39.535	NM	0.986	0.986	97.674	0.986	0.986	97.674			
3	CPM	0.901	0.9	93.023	1	1	100	NM	1	1	100	1	1	100			
4	CPM	0.536	0.535	51.163	0.542	0.289	23.256	NM	0.988	0.926	97.674	0.972	0.972	95.349			
0	ClusterOSS	0.958	0.957	93.023	0.972	0.972	95.349	OSS	0.929	0.926	97.674	0.986	0.986	97.674			
1	ClusterOSS	0.901	0.9	93.023	0.972	0.972	95.349	OSS	0.966	0.833	93.023	0.915	0.913	95.349			
2	ClusterOSS	1	1	100	0.944	0.943	90.698	OSS	0.994	0.926	97.674	1	1	100			
3	ClusterOSS	0.917	0.913	86.047	0.875	0.866	79.07	OSS	0.915	0.913	95.349	1	1	100			
4	ClusterOSS	1	1	100	0.986	0.986	97.674	OSS	0.901	0.9	93.023	0.929	0.926	97.674			
0	EE	0.958	0.957	93.023	0.986	0.986	97.674	RU	1	1	100	0.915	0.913	95.349			
1	EE	0.887	0.886	90.698	0.929	0.926	97.674	RU	0.915	0.913	95.349	0.929	0.926	97.674			
2	EE	1	1	100	1	1	100	RU	0.873	0.873	88.372	0.929	0.926	97.674			
3	EE	0.845	0.845	83.721	0.929	0.926	97.674	RU	0.931	0.928	88.372	1	1	100			
4	EE	0.929	0.926	97.674	0.929	0.926	97.674	RU	0.986	0.986	97.674	1	1	100			
0	ENN	0.992	0.913	95.349	0.786	0.756	93.023	SCB	1	0.986	97.674	0.857	0.845	95.349			
1	ENN	1	1	100	0.786	0.756	93.023	SCB	0.849	0.821	90.698	0.714	0.655	90.698			
2	ENN	0.786	0.756	93.023	0.714	0.655	90.698	SCB	0.913	0.9	93.023	0.786	0.756	93.023			
3	ENN	0.915	0.913	95.349	0.786	0.756	93.023	SCB	0.994	0.926	97.674	0.714	0.655	90.698			
4	ENN	0.986	0.986	97.674	0.857	0.845	95.349	SCB	0.843	0.833	93.023	0.857	0.845	95.349			
0	EUS	0.845	0.845	83.721	0.857	0.845	95.349	TL	0.929	0.926	97.674	0.786	0.756	93.023			
1	EUS	0.986	0.986	97.674	0.944	0.943	90.698	TL	0.917	0.913	95.349	0.786	0.756	93.023			
2	EUS	0.901	0.9	93.023	0.926	0.926	97.674	TL	0.889	0.886	90.698	0.786	0.756	93.023			
3	EUS	1	1	100	1	1	100	TL	0.79	0.845	95.349	0.714	0.655	90.698			
4	EUS	0.887	0.886	90.698	0.929	0.926	97.674	TL	0.917	0.913	95.349	0.714	0.655	90.698			
0	IHTS	0.931	0.928	88.372	0.819	0.799	69.767										
1	IHTS	0.958	0.957	93.023	0.944	0.943	90.698										
2	IHTS	0.929	0.926	97.674	0.708	0.645	51.163										
3	IHTS	1	1	100	1	1	100										
4	IHTS	0.817	0.816	79.07	0.929	0.926	97.674										
		new-thyroid2															
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%			
0	BC	0.986	0.986	97.674	1	1	100	IPADE-ID	0.5	0	16.279	0.653	0.553	41.86			
1	BC	0.944	0.943	90.698	0.915	0.913	95.349	IPADE-ID	0.5	0	16.279	0.786	0.756	93.023			
2	BC	1	1	100	0.986	0.986	97.674	IPADE-ID	0.5	0	16.279	0.5	0	83.721			
3	BC	0.859	0.859	86.047	0.929	0.926	97.674	IPADE-ID	0.772	0.745	90.698	0.5	0	83.721			
4	BC	0.815	0.809	88.372	1	1	100	IPADE-ID	0.73	0.713	83.721	0.889	0.882	81.395			
0	CNN	1	1	100	0.972	0.972	95.349	NCL	0.853	0.913	95.349	0.929	0.926	97.674			
1	CNN	0.788	0.784	83.721	0.986	0.986	97.674	NCL	1	1	100	0.786	0.756	93.023			
2	CNN	0.615	0.519	83.721	1	1	100	NCL	0.915	0.913	95.349	0.786	0.756	93.023			
3	CNN	0.774	0.772	81.395	0.931	0.928	88.372	NCL	0.903	0.913	95.349	0.643	0.535	88.372			
4	CNN	0.744	0.724	86.047	0.917	0.913	86.047	NCL	0.972	0.972	95.349	0.714	0.655	90.698			
0	CPM	0.901	0.9	93.023	1	1	100	NM	1	1	100	1	1	100			
1	CPM	0.931	0.928	88.372	0.986	0.986	97.674	NM	0.958	0.957	93.023	0.986	0.986	97.674			
2	CPM	0.758	0.735	88.372	0.915	0.913	95.349	NM	0.978	0.972	95.349	0.929	0.926	97.674			
3	CPM	0.786	0.756	93.023	1	1	100	NM	0.978	0.913	95.349	1	1	100			
4	CPM	0.944	0.943	90.698	1	1	100	NM	0.929	0.926	97.674	1	1	100			
0	ClusterOSS	0.986	0.986	97.674	0.986	0.986	97.674	OSS	0.972	0.972	95.349	1	1	100			
1	ClusterOSS	0.887	0.886	90.698	0.929	0.926	97.674	OSS	0.869	0.845	95.349	0.857	0.845	95.349			
2	ClusterOSS	0.819	0.799	69.767	0.986	0.986	97.674	OSS	0.972	0.972	95.349	0.786	0.756	93.023			
3	ClusterOSS	0.901	0.9	93.023	0.972	0.972	95.349	OSS	1	1	100	1	1	100			
4	ClusterOSS	0.915	0.913	95.349	0.929	0.926	97.674	OSS	0.929	0.926	97.674	0.714	0.655	90.698			
0	EE	0.944	0.943	90.698	1	1	100	RU	0.843	0.833	93.023	0.857	0.845	95.349			
1	EE	0.946	0.886	90.698	1	1	100	RU	0.903	0.898	83.721	0.929	0.926	97.674			
2	EE	0.887	0.886	90.698	0.929	0.926	97.674	RU	0.986	0.986	97.674	0.986	0.986	97.674			
3	EE	0.887	0.886	90.698	0.986	0.986	97.674	RU	1	1	100	1	1	100			
4	EE	0.968	0.972	95.349	0.929	0.926	97.674	RU	0.835	0.809	88.372	1	1	100			
0	ENN	0.762	0.756	93.023	0.714	0.655	90.698	SCB	0.972	0.972	95.349	0.643	0.535	88.372			
1	ENN	1	1	100	0.857	0.845	95.349	SCB	0.794	0.845	95.349	0.857	0.845	95.349			
2	ENN	0.98	0.845	95.349	0.714	0.655	90.698	SCB	0.829	0.821	90.698	0.714	0.655	90.698			
3	ENN	0.978	0.972	95.349	0.857	0.845	95.349	SCB	0.972	0.972	95.349	0.929	0.926	97.674			</

page-blocks0																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.962	0.949	92.968	0.83	0.828	87.945	IPADE-ID	0.5	0	10.228	0.5	0	10.228		
1	BC	0.943	0.944	94.155	0.868	0.865	92.694	IPADE-ID	0.5	0	10.228	0.5	0	10.228		
2	BC	0.922	0.917	92.237	0.848	0.844	91.963	IPADE-ID	0.5	0	10.228	0.5	0	10.228		
3	BC	0.978	0.951	93.333	0.847	0.844	90.228	IPADE-ID	0.5	0	10.228	0.5	0	10.228		
4	BC	0.965	0.949	94.414	0.856	0.855	89.286	IPADE-ID	0.5	0	10.165	0.5	0	10.165		
0	CNN	0.893	0.901	96.347	0.567	0.366	91.142	NCL	0.963	0.943	96.895	0.76	0.726	93.79		
1	CNN	0.895	0.902	93.151	0.584	0.411	91.324	NCL	0.912	0.931	97.078	0.75	0.71	94.247		
2	CNN	0.943	0.906	96.438	0.668	0.582	92.968	NCL	0.926	0.928	97.352	0.733	0.685	93.973		
3	CNN	0.92	0.875	94.612	0.558	0.341	90.959	NCL	0.947	0.881	96.621	0.714	0.658	93.333		
4	CNN	0.926	0.92	92.857	0.765	0.737	92.949	NCL	0.881	0.89	95.971	0.709	0.649	93.681		
0	CPM	0.899	0.844	86.301	0.736	0.691	93.881	NM	0.866	0.801	71.598	0.727	0.727	70.959		
1	CPM	0.929	0.932	94.886	0.645	0.547	91.689	NM	0.913	0.911	89.132	0.748	0.745	69.041		
2	CPM	0.93	0.901	95.434	0.786	0.764	93.607	NM	0.883	0.8	70.868	0.717	0.715	67.032		
3	CPM	0.955	0.912	95.068	0.585	0.412	91.507	NM	0.656	0.685	55.982	0.748	0.746	70.32		
4	CPM	0.954	0.933	95.238	0.61	0.473	91.667	NM	0.662	0.677	53.114	0.756	0.753	70.513		
0	ClusterOSS	0.665	0.64	47.215	0.5	0	10.228	OSS	0.936	0.902	96.53	0.748	0.706	94.521		
1	ClusterOSS	0.637	0.61	43.836	0.5	0	10.228	OSS	0.975	0.924	97.443	0.678	0.597	93.242		
2	ClusterOSS	0.803	0.778	64.932	0.5	0	10.228	OSS	0.952	0.893	96.347	0.707	0.65	92.785		
3	ClusterOSS	0.894	0.806	69.498	0.5	0	10.228	OSS	0.91	0.927	97.078	0.757	0.718	94.703		
4	ClusterOSS	0.71	0.633	46.612	0.5	0	10.165	OSS	0.936	0.907	96.795	0.689	0.615	93.681		
0	EE	0.965	0.95	93.881	0.859	0.857	91.05	RU	0.947	0.926	91.781	0.853	0.851	89.863		
1	EE	0.961	0.958	94.612	0.833	0.826	91.233	RU	0.922	0.938	94.612	0.825	0.819	90.594		
2	EE	0.955	0.944	95.708	0.868	0.866	91.233	RU	0.981	0.969	96.53	0.873	0.872	91.416		
3	EE	0.965	0.952	94.886	0.829	0.826	88.493	RU	0.959	0.934	93.151	0.833	0.829	89.863		
4	EE	0.967	0.937	92.949	0.854	0.851	91.026	RU	0.95	0.941	91.575	0.865	0.865	89.469		
0	ENN	0.938	0.901	97.169	0.742	0.702	93.516	SCB	0.978	0.966	94.977	0.824	0.809	94.612		
1	ENN	0.97	0.945	97.352	0.761	0.727	94.064	SCB	0.973	0.949	95.16	0.78	0.758	92.42		
2	ENN	0.955	0.931	97.078	0.743	0.702	93.607	SCB	0.953	0.942	95.982	0.803	0.787	93.059		
3	ENN	0.958	0.945	97.26	0.729	0.683	93.333	SCB	0.942	0.932	93.516	0.825	0.812	94.155		
4	ENN	0.932	0.919	97.344	0.737	0.69	94.414	SCB	0.948	0.939	94.048	0.779	0.757	92.674		
0	EUS	0.926	0.922	92.329	0.826	0.821	90.137	TL	0.972	0.921	96.712	0.627	0.508	92.055		
1	EUS	0.945	0.935	93.242	0.852	0.851	89.132	TL	0.929	0.929	97.534	0.785	0.758	94.703		
2	EUS	0.952	0.948	94.155	0.846	0.841	91.507	TL	0.94	0.924	96.621	0.678	0.597	93.333		
3	EUS	0.966	0.958	94.612	0.886	0.885	92.329	TL	0.961	0.937	97.443	0.704	0.64	93.699		
4	EUS	0.953	0.931	94.048	0.866	0.864	90.934	TL	0.957	0.896	97.253	0.713	0.655	93.59		
pima																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.699	0.647	62.987	0.784	0.784	78.571	IPADE-ID	0.501	0.14	35.714	0.5	0	35.065		
1	BC	0.74	0.735	73.377	0.799	0.799	80.519	IPADE-ID	0.496	0.099	35.065	0.5	0	35.065		
2	BC	0.797	0.744	71.429	0.765	0.765	76.623	IPADE-ID	0.5	0	35.065	0.5	0	35.065		
3	BC	0.706	0.63	62.987	0.69	0.687	70.779	IPADE-ID	0.496	0.17	35.714	0.5	0	35.065		
4	BC	0.742	0.729	71.711	0.678	0.675	69.737	IPADE-ID	0.505	0	34.211	0.5	0	34.211		
0	CNN	0.729	0.599	71.429	0.714	0.691	76.623	NCL	0.724	0.727	71.429	0.772	0.769	79.221		
1	CNN	0.614	0.514	71.429	0.692	0.646	76.623	NCL	0.679	0.697	70.13	0.726	0.714	76.623		
2	CNN	0.761	0.664	72.078	0.666	0.64	72.078	NCL	0.696	0.667	69.481	0.705	0.701	72.727		
3	CNN	0.715	0.74	75.974	0.723	0.703	77.273	NCL	0.725	0.76	74.675	0.768	0.762	79.87		
4	CNN	0.65	0.599	63.158	0.676	0.634	75	NCL	0.748	0.69	70.395	0.718	0.71	75		
0	CPM	0.656	0.567	65.584	0.614	0.572	68.182	NM	0.737	0.697	69.481	0.737	0.736	74.675		
1	CPM	0.636	0.659	67.532	0.71	0.678	77.273	NM	0.731	0.693	69.481	0.715	0.715	70.779		
2	CPM	0.577	0.464	70.779	0.711	0.673	77.922	NM	0.778	0.736	74.675	0.735	0.734	72.727		
3	CPM	0.751	0.699	73.377	0.773	0.766	80.519	NM	0.688	0.672	65.584	0.743	0.74	75.974		
4	CPM	0.774	0.637	77.632	0.743	0.728	78.947	NM	0.754	0.745	74.342	0.745	0.745	75		
0	ClusterOSS	0.758	0.721	68.831	0.687	0.671	64.286	OSS	0.671	0.653	67.532	0.691	0.653	75.974		
1	ClusterOSS	0.547	0.514	50	0.5	0	35.065	OSS	0.808	0.793	79.221	0.763	0.757	79.221		
2	ClusterOSS	0.437	0.454	48.052	0.497	0.261	62.338	OSS	0.757	0.663	77.922	0.718	0.699	76.623		
3	ClusterOSS	0.539	0.524	50	0.519	0.518	51.299	OSS	0.708	0.699	75.325	0.719	0.691	77.922		
4	ClusterOSS	0.607	0.593	59.211	0.667	0.667	67.763	OSS	0.708	0.678	71.053	0.704	0.684	75.658		
0	EE	0.803	0.781	75.974	0.775	0.775	78.571	RU	0.664	0.654	65.584	0.708	0.707	72.078		
1	EE	0.605	0.61	62.987	0.67	0.669	68.182	RU	0.807	0.731	73.377	0.778	0.778	77.273		
2	EE	0.645	0.664	67.532	0.743	0.74	75.974	RU	0.679	0.676	66.883	0.758	0.753	78.571		
3	EE	0.772	0.731	73.377	0.743	0.74	75.974	RU	0.73	0.652	68.182	0.696	0.69	72.078		
4	EE	0.74	0.715	68.421	0.764	0.764	75.658	RU	0.724	0.725	70.395	0.731	0.731	73.684		
0	ENN	0.824	0.764	78.571	0.763	0.759	78.571	SCB	0.662	0.649	64.935	0.741	0.741	74.675		
1	ENN	0.674	0.695	72.727	0.74	0.732	77.273	SCB	0.747	0.71	70.13	0.787	0.787	77.922		
2	ENN	0.761	0.755	75.974	0.743	0.739	76.623	SCB	0.729	0.683	66.234	0.707	0.707	70.779		
3	ENN	0.74	0.697	71.429	0.645	0.628	68.831	SCB	0.687	0.685	66.234	0.755	0.746	72.078		
4	ENN	0.774	0.709	69.737	0.712	0.71	73.026	SCB	0.801	0.784	75.658	0.769	0.768	75.658		
0	EUS	0.802	0.672	74.675	0.725	0.719	75.325	TL	0.695	0.644	70.13	0.691	0.658	75.325		
1	EUS	0.678	0.655	63.636	0.697	0.697	69.481	TL	0.58	0.604	68.831	0.729	0.698	79.221		
2	EUS	0.79	0.712	72.078	0.771	0.77	78.571	TL	0.707	0.682	72.727	0.751	0.737	79.221		
3	EUS	0.722	0.729	72.078	0.777	0.773	79.87	TL	0.685	0.627	71.429	0.7	0.671	75.974		
4	EUS	0.635	0.615	58.553	0.72	0.718	70.395	TL	0.768	0.689	67.763	0.69	0.668	74.342		
0	IHTS	0.5	0	35.065	0.5	0	35.065									
1	IHTS	0.5	0	35.065	0.5	0	35.065</td									

segment0																		
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%				
0	BC	0.991	0.991	98.485	0.987	0.987	97.835	IPADE-ID	0.462	0	85.714	0.624	0.516	87.446				
1	BC	0.977	0.977	97.186	0.979	0.979	97.403	IPADE-ID	0.5	0	85.714	0.612	0.51	85.498				
2	BC	0.958	0.958	97.186	0.991	0.991	98.485	IPADE-ID	0.75	0.707	92.857	0.598	0.598	61.472				
3	BC	0.981	0.979	97.403	0.98	0.98	97.619	IPADE-ID	0.48	0	85.714	0.65	0.649	61.688				
4	BC	0.981	0.975	99.13	0.952	0.952	95	IPADE-ID	0.5	0	85.87	0.5	0	85.87				
0	CNN	0.912	0.938	97.186	0.947	0.945	98.485	NCL	0.982	0.976	99.134	0.977	0.977	99.351				
1	CNN	0.975	0.973	98.701	0.5	0	85.714	NCL	0.987	0.987	98.918	0.992	0.992	99.784				
2	CNN	0.977	0.971	97.186	0.878	0.869	96.32	NCL	0.998	0.999	99.784	1	1	100				
3	CNN	0.908	0.907	93.939	0.54	0.322	85.065	NCL	0.984	0.983	99.351	0.991	0.991	99.567				
4	CNN	0.944	0.944	93.696	0.506	0.124	85.87	NCL	0.965	0.96	97.609	0.991	0.991	99.565				
0	CPM	0.965	0.965	96.104	0.538	0.275	86.797	NM	0.96	0.959	93.074	0.946	0.945	92.857				
1	CPM	0.964	0.928	90.043	0.97	0.97	98.052	NM	0.958	0.953	94.156	0.96	0.959	94.156				
2	CPM	0.942	0.941	97.619	0.955	0.953	98.701	NM	0.986	0.982	96.97	0.965	0.964	93.939				
3	CPM	0.884	0.882	91.991	0.968	0.968	98.918	NM	0.904	0.882	81.818	0.958	0.958	93.939				
4	CPM	0.742	0.728	92.174	0.5	0	85.87	NM	0.94	0.962	94.565	0.933	0.931	89.565				
0	ClusterOSS	0.947	0.958	95.022	0.958	0.958	93.939	OSS	0.982	0.976	99.134	0.992	0.992	99.784				
1	ClusterOSS	0.961	0.958	95.022	0.785	0.755	63.203	OSS	0.99	0.994	98.918	1	1	100				
2	ClusterOSS	0.912	0.907	84.848	0.774	0.74	61.255	OSS	0.992	0.99	99.351	0.992	0.992	99.784				
3	ClusterOSS	0.963	0.963	95.887	0.867	0.867	84.848	OSS	0.985	0.983	99.351	0.982	0.982	99.134				
4	ClusterOSS	0.5	0	14.13	0.699	0.63	48.261	OSS	0.966	0.977	99.348	0.992	0.992	99.785				
0	EE	0.984	0.985	98.485	0.971	0.971	96.104	RU	0.972	0.968	97.835	0.98	0.98	98.701				
1	EE	0.965	0.971	97.186	0.992	0.992	98.701	RU	0.996	0.991	98.485	0.972	0.972	95.238				
2	EE	0.976	0.976	98.052	0.98	0.98	97.619	RU	0.992	0.992	98.701	0.994	0.994	98.918				
3	EE	0.985	0.989	98.052	0.968	0.968	96.753	RU	0.963	0.968	97.835	0.997	0.997	99.567				
4	EE	0.998	0.995	99.13	0.986	0.986	98.696	RU	0.961	0.961	97.826	0.95	0.95	94.783				
0	ENN	0.991	0.992	99.784	0.992	0.992	99.784	SCB	0.997	0.989	99.134	0.997	0.997	99.567				
1	ENN	0.971	0.975	98.918	0.991	0.991	99.567	SCB	0.998	0.992	99.784	0.996	0.996	99.351				
2	ENN	0.992	0.992	99.784	0.992	0.992	99.784	SCB	0.981	0.976	98.052	0.992	0.992	99.784				
3	ENN	0.997	0.997	99.567	0.999	0.999	99.784	SCB	0.963	0.967	98.701	0.98	0.98	98.701				
4	ENN	0.975	0.96	98.696	0.977	0.977	99.348	SCB	0.963	0.958	98.478	0.981	0.981	98.913				
0	EUS	0.982	0.982	96.967	0.991	0.991	98.485	TL	0.999	0.999	99.784	1	1	100				
1	EUS	0.959	0.958	98.268	0.958	0.958	96.104	TL	0.968	0.968	98.918	0.976	0.976	99.134				
2	EUS	0.984	0.99	98.268	0.977	0.977	97.186	TL	0.999	0.997	99.567	0.991	0.991	99.567				
3	EUS	0.974	0.977	98.268	0.98	0.98	97.619	TL	0.968	0.968	98.918	0.992	0.992	99.784				
4	EUS	0.992	0.989	99.13	0.986	0.986	98.696	TL	0.998	0.999	99.783	0.992	0.992	99.783				
0	IHTS	0.991	0.994	98.918	0.975	0.975	97.835											
1	IHTS	0.9	0.537	38.961	0.821	0.801	69.264											
2	IHTS	0.995	0.995	99.134	1	1	100											
3	IHTS	0.777	0.748	62.771	0.801	0.775	65.801											
4	IHTS	0.713	0.748	62.174	0.816	0.796	68.478											
vehicle0																		
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%				
0	BC	0.937	0.925	95.294	0.965	0.965	94.706	IPADE-ID	0.5	0	76.471	0.5	0	76.471				
1	BC	0.963	0.945	92.941	0.95	0.949	92.353	IPADE-ID	0.5	0	23.529	0.5	0	76.471				
2	BC	0.933	0.929	91.765	0.946	0.945	91.765	IPADE-ID	0.5	0	76.471	0.5	0	76.471				
3	BC	0.916	0.933	92.353	0.938	0.936	90.583	IPADE-ID	0.5	0	76.471	0.5	0	76.471				
4	BC	0.94	0.922	92.169	0.952	0.951	93.976	IPADE-ID	0.5	0	76.506	0.5	0	76.506				
0	CNN	0.85	0.849	89.412	0.907	0.906	92.353	NCL	0.9	0.894	93.529	0.942	0.941	96.471				
1	CNN	0.851	0.778	86.471	0.897	0.894	93.529	NCL	0.95	0.943	95.294	0.902	0.9	92.941				
2	CNN	0.881	0.891	92.941	0.892	0.888	94.118	NCL	0.952	0.929	91.765	0.936	0.936	94.118				
3	CNN	0.92	0.9	92.941	0.913	0.908	95.882	NCL	0.927	0.874	91.765	0.951	0.951	96.471				
4	CNN	0.888	0.881	91.566	0.89	0.885	93.976	NCL	0.933	0.921	93.373	0.964	0.963	95.783				
0	CPM	0.977	0.932	93.529	0.942	0.941	96.471	NM	0.732	0.699	66.471	0.968	0.968	96.471				
1	CPM	0.893	0.881	92.941	0.892	0.888	94.118	NM	0.756	0.769	70	0.907	0.906	92.353				
2	CPM	0.917	0.887	89.412	0.913	0.911	94.706	NM	0.923	0.932	93.529	0.963	0.963	97.059				
3	CPM	0.972	0.887	92.353	0.939	0.939	94.706	NM	0.7	0.74	67.647	0.963	0.963	97.059				
4	CPM	0.934	0.9	93.373	0.964	0.963	95.783	NM	0.618	0.743	67.47	0.98	0.98	96.988				
0	ClusterOSS	0.79	0.745	66.471	0.796	0.77	68.824	OSS	0.937	0.849	91.176	0.955	0.954	97.059				
1	ClusterOSS	0.788	0.672	73.529	0.607	0.603	57.059	OSS	0.963	0.968	96.471	0.934	0.932	96.471				
2	ClusterOSS	0.922	0.921	89.412	0.762	0.723	63.529	OSS	0.921	0.927	94.118	0.968	0.968	96.471				
3	ClusterOSS	0.915	0.875	82.941	0.894	0.78	70	OSS	0.929	0.884	91.765	0.961	0.96	95.294				
4	ClusterOSS	0.893	0.86	80.12	0.728	0.676	58.434	OSS	0.924	0.863	92.169	0.903	0.902	93.373				
0	EE	0.884	87.647	0.946	0.945	91.765	RU	0.938	0.913	89.412	0.954	0.953	92.941					
1	EE	0.886	0.891	90	0.949	0.949	93.529	RU	0.923	0.869	85.294	0.942	0.941	91.176				
2	EE	0.937	0.915	92.353	0.962	0.961	94.118	RU	0.944	0.91	92.941	0.977	0.977	96.471				
3	EE	0.954	0.953	94.118	0.946	0.945	91.765	RU	0.915	0.91	92.941	0.973	0.973	95.882				
4	EE	0.926	0.895	93.976	0.956	0.956	94.578	RU	0.93	0.93	93.373	0.928	0.927	90.361				
0	ENN	0.961	0.															

vehicle1																				
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%						
0	BC	0.718	0.728	71.765	0.754	0.752	72.353	IPADE-ID	0.5	0	74.118	0.5	0	74.118						
1	BC	0.768	0.764	78.235	0.721	0.721	72.941	IPADE-ID	0.5	0	74.118	0.5	0	74.118						
2	BC	0.713	0.686	68.824	0.69	0.689	68.235	IPADE-ID	0.5	0	25.882	0.604	0.492	43.529						
3	BC	0.752	0.682	68.235	0.658	0.658	64.706	IPADE-ID	0.5	0	25.882	0.5	0	74.118						
4	BC	0.703	0.729	70.482	0.742	0.742	73.494	IPADE-ID	0.5	0	24.699	0.5	0	75.301						
0	CNN	0.701	0.702	66.471	0.511	0.151	74.706	NCL	0.665	0.562	70.588	0.598	0.45	78.824						
1	CNN	0.726	0.541	72.353	0.5	0	74.118	NCL	0.734	0.732	78.824	0.598	0.45	78.824						
2	CNN	0.641	0.426	78.824	0.541	0.3	75.882	NCL	0.697	0.655	72.353	0.619	0.541	76.471						
3	CNN	0.545	0.365	75.882	0.5	0	74.118	NCL	0.641	0.573	72.941	0.617	0.496	79.412						
4	CNN	0.707	0.634	74.096	0.5	0	75.301	NCL	0.697	0.716	76.506	0.651	0.559	81.928						
0	CPM	0.736	0.674	72.353	0.5	0	74.118	NM	0.625	0.55	48.824	0.682	0.682	68.235						
1	CPM	0.756	0.501	75.294	0.568	0.369	77.647	NM	0.743	0.688	64.706	0.734	0.73	69.412						
2	CPM	0.642	0.628	73.529	0.5	0	74.118	NM	0.63	0.55	72.941	0.726	0.724	74.706						
3	CPM	0.62	0.414	74.706	0.511	0.151	74.706	NM	0.663	0.532	67.059	0.707	0.704	74.118						
4	CPM	0.672	0.628	72.892	0.63	0.532	80.12	NM	0.705	0.665	62.651	0.689	0.689	69.277						
0	ClusterOSS	0.688	0.666	59.412	0.683	0.674	62.941	OSS	0.798	0.72	82.353	0.511	0.151	74.706						
1	ClusterOSS	0.442	0.416	62.353	0.5	0	74.118	OSS	0.749	0.721	77.059	0.523	0.213	75.294						
2	ClusterOSS	0.536	0.415	36.471	0.654	0.652	62.941	OSS	0.708	0.546	73.529	0.5	0	74.118						
3	ClusterOSS	0.629	0.573	56.471	0.62	0.602	54.706	OSS	0.622	0.601	71.176	0.534	0.261	75.882						
4	ClusterOSS	0.463	0.426	46.386	0.543	0.528	60.843	OSS	0.683	0.601	75.904	0.512	0.156	75.904						
0	EE	0.71	0.715	70	0.724	0.724	71.176	RU	0.726	0.722	67.059	0.686	0.685	67.647						
1	EE	0.762	0.74	67.647	0.733	0.722	67.059	RU	0.773	0.758	74.118	0.712	0.711	69.412						
2	EE	0.756	0.73	69.412	0.695	0.693	72.353	RU	0.748	0.722	70	0.652	0.651	65.882						
3	EE	0.78	0.737	74.118	0.697	0.696	68.235	RU	0.764	0.733	73.529	0.778	0.777	75.882						
4	EE	0.832	0.762	72.289	0.794	0.794	80.12	RU	0.747	0.724	74.699	0.734	0.734	73.494						
0	ENN	0.721	0.733	74.706	0.69	0.664	78.235	SCB	0.679	0.703	72.941	0.643	0.627	71.176						
1	ENN	0.73	0.681	67.059	0.651	0.572	80	SCB	0.737	0.747	71.765	0.721	0.721	71.765						
2	ENN	0.724	0.711	75.294	0.688	0.639	81.176	SCB	0.737	0.741	71.765	0.746	0.741	78.824						
3	ENN	0.722	0.72	75.294	0.668	0.625	78.235	SCB	0.758	0.788	73.529	0.75	0.747	78.235						
4	ENN	0.727	0.724	74.699	0.639	0.578	77.711	SCB	0.72	0.685	69.88	0.713	0.705	76.506						
0	EUS	0.775	0.771	74.118	0.713	0.713	71.765	TL	0.715	0.658	72.941	0.591	0.426	78.824						
1	EUS	0.712	0.733	74.706	0.763	0.763	75.882	TL	0.693	0.633	77.059	0.5	0	74.118						
2	EUS	0.718	0.737	74.118	0.754	0.752	72.353	TL	0.777	0.652	75.882	0.553	0.336	76.471						
3	EUS	0.765	0.711	69.412	0.695	0.694	71.176	TL	0.732	0.712	77.059	0.5	0	74.118						
4	EUS	0.749	0.705	71.687	0.71	0.709	68.675	TL	0.728	0.72	74.096	0.545	0.311	77.108						
0	IHTS	0.676	0.737	71.176	0.702	0.683	62.353													
1	IHTS	0.5	0	25.882	0.5	0	25.882													
2	IHTS	0.654	0.614	54.706	0.504	0.089	26.471													
3	IHTS	0.727	0.703	65.882	0.681	0.677	64.706													
4	IHTS	0.702	0.673	65.663	0.746	0.744	71.687													
vehicle2																				
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%						
0	BC	0.941	0.942	94.706	0.89	0.888	85.882	IPADE-ID	0.5	0	74.118	0.5	0	25.882						
1	BC	0.933	0.934	93.529	0.897	0.891	84.706	IPADE-ID	0.5	0	74.118	0.5	0	25.882						
2	BC	0.965	0.934	90.588	0.946	0.945	94.118	IPADE-ID	0.5	0	25.882	0.5	0	25.882						
3	BC	0.964	0.961	95.294	0.876	0.876	87.059	IPADE-ID	0.5	0	25.882	0.508	0.126	27.059						
4	BC	0.953	0.944	93.976	0.92	0.92	91.566	IPADE-ID	0.5	0	74.699	0.5	0	25.301						
0	CNN	0.946	0.942	94.706	0.659	0.564	82.353	NCL	0.924	0.907	92.941	0.874	0.869	92.353						
1	CNN	0.837	0.788	88.824	0.598	0.45	78.824	NCL	0.997	0.996	99.412	0.908	0.907	92.941						
2	CNN	0.916	0.892	90.588	0.761	0.73	86.471	NCL	0.948	0.93	95.294	0.856	0.846	91.765						
3	CNN	0.814	0.798	82.353	0.708	0.652	84.118	NCL	0.952	0.965	95.882	0.915	0.915	92.941						
4	CNN	0.931	0.919	93.976	0.56	0.345	77.711	NCL	0.979	0.964	95.783	0.889	0.885	92.771						
0	CPM	0.896	0.915	92.941	0.705	0.64	84.706	NM	0.949	0.949	94.706	0.922	0.921	90.588						
1	CPM	0.912	0.911	93.529	0.598	0.45	78.824	NM	0.935	0.926	92.353	0.914	0.913	89.412						
2	CPM	0.915	0.894	93.529	0.746	0.704	86.471	NM	0.985	0.988	98.235	0.888	0.888	90						
3	CPM	0.927	0.868	87.059	0.779	0.761	85.882	NM	0.88	0.922	95.294	0.888	0.888	90						
4	CPM	0.885	0.883	90.964	0.754	0.725	85.542	NM	0.933	0.938	93.976	0.896	0.896	90.361						
0	ClusterOSS	0.741	0.772	70	0.508	0.126	27.059	OSS	0.967	0.969	97.647	0.893	0.893	92.941						
1	ClusterOSS	0.855	0.733	84.706	0.497	0.495	51.765	OSS	0.939	0.885	93.529	0.864	0.853	92.941						
2	ClusterOSS	0.841	0.839	79.412	0.512	0.216	28.824	OSS	0.96	0.957	94.706	0.851	0.847	88.824						
3	ClusterOSS	0.777	0.745	67.059	0.758	0.718	64.118	OSS	0.94	0.95	95.882	0.894	0.888	94.118						
4	ClusterOSS	0.881	0.895	87.952	0.847	0.841	79.518	OSS	0.92	0.919	93.976	0.876	0.875	89.759						
0	EE	0.921	0.907	92.941	0.879	0.879	86.471	RU	0.957	0.941	93.529	0.906	0.905	88.235						
1	EE	0.989	0.961	96.471	0.949	0.949	93.529	RU	0.933	0.941	93.529	0.942	0.941	93.529						
2	EE	0.909	0.923	92.941	0.925	0.924	90	RU	0.904	0.906	89.412	0.902	0.9	87.647						
3	EE	0.944	0.915	91.765	0.913	0.911	88.235	RU	0.902	0.903	91.176	0.914	0.914	90.588						
4	EE	0.971	0.96	96.386	0.876	0.875	86.145	RU	0.949	0.936	92.771	0.908	0.908	89.759						
0	ENN	0.894	0.907	92.941	0.927	0.926	94.706	SCB	0.962	0.965	95.882	0.968	0.968	95.294						
1	ENN	0.936	0.923	94.118	0.836	0.826	90	SCB	0.934	0.919	93.529	0.911	0.911	91.176						
2	ENN	0.975	0.969	96.471	0.934	0.934	93.529	SCB	0.908	0.907	92.941	0.878	0.874	91.765						
3	ENN	0.963	0.922	95.294	0.852	0.843	91.176	SCB	0.981	0.981	98.235	0.9	0.899	90.588						
4	ENN	0.976	0.96	96.386	0.96	0.96	96.386	SCB	0.94	0.927	95.181	0.908	0.907	93.373						
0	EUS	0.956	0.949	94.706	0.922	0.921	90.588	TL	0.975	0.961	97.647	0.863	0.856</							

vehicle3															
Fold	Algoritmo	j48auc	j48gm	j48-%	SVMauc	SVMgm	SVM-%	Algoritmo	j48auc	j48gm	j48-%	SVMauc	SVMgm	SVM-%	
0	BC	0.763	0.744	72.353	0.784	0.775	72.353	IPADE-ID	0.5	0	74.706	0.615	0.528	45.882	
1	BC	0.71	0.721	74.706	0.637	0.632	67.647	IPADE-ID	0.5	0	74.706	0.657	0.65	70.588	
2	BC	0.813	0.736	77.059	0.703	0.702	68.235	IPADE-ID	0.5	0	74.706	0.59	0.585	62.941	
3	BC	0.787	0.735	69.412	0.738	0.734	70	IPADE-ID	0.5	0	74.706	0.528	0.342	32.941	
4	BC	0.735	0.719	69.277	0.754	0.75	71.687	IPADE-ID	0.5	0	75.904	0.564	0.431	75.301	
0	CNN	0.654	0.519	74.118	0.546	0.337	75.882	NCL	0.797	0.751	81.176	0.512	0.152	75.294	
1	CNN	0.667	0.583	74.118	0.57	0.374	78.235	NCL	0.705	0.601	67.647	0.5	0	74.706	
2	CNN	0.651	0.568	74.118	0.5	0	74.706	NCL	0.835	0.784	77.059	0.5	0	74.706	
3	CNN	0.688	0.519	70	0.5	0	74.706	NCL	0.707	0.609	67.059	0.5	0	74.706	
4	CNN	0.658	0.577	76.506	0.5	0	55.904	NCL	0.748	0.7	76.506	0.581	0.478	75.301	
0	CPM	0.763	0.705	73.529	0.5	0	74.706	NM	0.673	0.563	49.412	0.692	0.691	70	
1	CPM	0.576	0.371	77.059	0.512	0.152	75.294	NM	0.745	0.715	72.353	0.68	0.678	70.588	
2	CPM	0.695	0.631	57.647	0.519	0.215	75.294	NM	0.686	0.644	65.294	0.63	0.614	70	
3	CPM	0.629	0.58	70.588	0.5	0	74.706	NM	0.691	0.585	64.706	0.739	0.736	77.059	
4	CPM	0.686	0.672	61.446	0.5	0	55.904	NM	0.502	0.5	54.819	0.702	0.7	72.892	
0	ClusterOSS	0.766	0.732	67.647	0.679	0.673	63.529	OSS	0.801	0.583	77.647	0.5	0	74.706	
1	ClusterOSS	0.688	0.638	57.059	0.587	0.438	39.412	OSS	0.743	0.702	80	0.5	0	74.706	
2	ClusterOSS	0.621	0.645	60.588	0.707	0.706	68.824	OSS	0.719	0.641	75.294	0.5	0	74.706	
3	ClusterOSS	0.529	0.504	51.176	0.43	0.426	45.882	OSS	0.637	0.565	67.647	0.5	0	74.706	
4	ClusterOSS	0.435	0.423	56.024	0.561	0.537	64.458	OSS	0.723	0.718	75.904	0.5	0	75.904	
0	EE	0.737	0.711	70.588	0.688	0.687	69.412	RU	0.674	0.675	66.471	0.691	0.691	68.824	
1	EE	0.807	0.776	75.882	0.738	0.736	71.176	RU	0.717	0.691	71.176	0.73	0.725	68.824	
2	EE	0.769	0.746	73.529	0.683	0.68	65.294	RU	0.727	0.681	61.176	0.707	0.707	70	
3	EE	0.812	0.797	75.294	0.746	0.742	71.176	RU	0.814	0.757	74.118	0.827	0.817	76.471	
4	EE	0.702	0.712	70.482	0.724	0.724	71.084	RU	0.766	0.755	70.482	0.683	0.683	67.47	
0	ENN	0.737	0.674	77.059	0.588	0.501	74.118	SCB	0.719	0.729	71.176	0.684	0.682	71.176	
1	ENN	0.772	0.737	75.882	0.531	0.263	75.882	SCB	0.68	0.658	68.824	0.7	0.696	73.529	
2	ENN	0.765	0.705	73.529	0.5	0	74.706	SCB	0.773	0.746	75.882	0.5	0	74.706	
3	ENN	0.714	0.671	76.471	0.5	0	74.706	SCB	0.822	0.742	75.294	0.684	0.68	72.353	
4	ENN	0.695	0.672	69.88	0.554	0.463	71.084	SCB	0.707	0.697	66.265	0.683	0.683	68.675	
0	EUS	0.818	0.8	76.471	0.715	0.714	70	TL	0.645	0.659	74.118	0.5	0	74.706	
1	EUS	0.544	0.597	61.765	0.722	0.722	71.176	TL	0.73	0.66	76.471	0.5	0	74.706	
2	EUS	0.748	0.697	65.882	0.73	0.729	71.176	TL	0.584	0.412	65.882	0.5	0	74.706	
3	EUS	0.77	0.773	77.647	0.648	0.648	63.529	TL	0.745	0.659	78.824	0.5	0	74.706	
4	EUS	0.747	0.74	72.289	0.729	0.727	70.482	TL	0.683	0.589	72.289	0.5	0	75.904	
0	IHTS	0.799	0.761	74.706	0.711	0.711	71.765								
1	IHTS	0.519	0.212	27.647	0.512	0.154	27.059								
2	IHTS	0.778	0.717	65.294	0.673	0.601	52.353								
3	IHTS	0.696	0.693	62.941	0.733	0.718	65.882								
4	IHTS	0.587	0.393	34.94	0.5	0	24.096								
wisconsin															
Fold	Algoritmo	j48auc	j48gm	j48-%	SVMauc	SVMgm	SVM-%	Algoritmo	j48auc	j48gm	j48-%	SVMauc	SVMgm	SVM-%	
0	BC	0.957	0.946	94.891	0.973	0.973	97.08	IPADE-ID	0.5	0	35.036	0.5	0	35.036	
1	BC	0.965	0.963	97.08	0.968	0.968	97.08	IPADE-ID	0.5	0	35.036	0.5	0	35.036	
2	BC	0.934	0.946	94.891	0.969	0.968	97.81	IPADE-ID	0.5	0	35.036	0.5	0	35.036	
3	BC	0.98	0.978	97.81	0.978	0.978	97.81	IPADE-ID	0.5	0	35.036	0.5	0	35.036	
4	BC	0.928	0.921	91.111	0.955	0.953	94.074	IPADE-ID	0.5	0	34.815	0.5	0	34.815	
0	CNN	0.951	0.957	96.35	0.958	0.957	96.35	NCL	0.979	0.973	97.08	0.989	0.989	98.54	
1	CNN	0.941	0.94	93.431	0.973	0.973	97.08	NCL	0.981	0.941	94.891	0.973	0.973	97.08	
2	CNN	0.92	0.914	89.781	0.962	0.962	96.35	NCL	0.93	0.898	91.241	0.957	0.957	95.62	
3	CNN	0.899	0.865	87.591	0.989	0.989	98.54	NCL	0.884	0.9	93.431	0.989	0.989	98.54	
4	CNN	0.93	0.863	84.444	0.962	0.962	96.296	NCL	0.957	0.957	96.296	0.957	0.957	96.296	
0	CPM	0.942	0.895	88.321	0.978	0.978	97.81	NM	0.942	0.946	94.891	0.952	0.952	95.62	
1	CPM	0.897	0.771	75.182	0.936	0.936	94.161	NM	0.975	0.952	96.35	0.984	0.984	98.54	
2	CPM	0.943	0.885	91.971	0.958	0.957	96.35	NM	0.941	0.946	94.891	0.968	0.968	97.08	
3	CPM	0.953	0.967	96.35	0.941	0.941	94.891	NM	0.932	0.951	94.891	0.961	0.961	95.62	
4	CPM	0.949	0.923	94.074	0.921	0.919	90.37	NM	0.913	0.891	90.37	0.989	0.989	98.519	
0	ClusterOSS	0.5	0	35.036	0.5	0	35.036	OSS	0.98	0.984	98.54	0.974	0.973	97.81	
1	ClusterOSS	0.5	0	35.036	0.5	0	35.036	OSS	0.94	0.947	95.62	0.958	0.957	96.35	
2	ClusterOSS	0.5	0	35.036	0.5	0	35.036	OSS	0.931	0.92	92.701	0.961	0.961	95.62	
3	ClusterOSS	0.5	0	35.036	0.5	0	35.036	OSS	0.947	0.962	96.35	0.968	0.968	97.08	
4	ClusterOSS	0.5	0	34.815	0.5	0	34.815	OSS	0.924	0.934	93.333	0.962	0.962	96.296	
0	EE	0.947	0.962	96.35	0.984	0.984	98.54	RU	0.896	0.898	91.241	0.967	0.967	96.35	
1	EE	0.941	0.929	92.701	0.957	0.957	95.62	RU	0.93	0.94	93.431	0.989	0.989	98.54	
2	EE	0.932	0.903	90.511	0.957	0.957	95.62	RU	0.944	0.941	94.161	0.967	0.967	96.35	
3	EE	0.919	0.936	94.891	0.978	0.977	97.08	RU	0.948	0.951	94.891	0.983	0.983	97.81	
4	EE	0.935	0.929	94.074	0.989	0.989	99.259	RU	0.931	0.946	95.556	0.957	0.957	96.296	
0	ENN	0.975	0.972	96.35	0.967	0.967	96.35	SCB	0.965	0.957	96.35	0.952	0.952	95.62	
1	ENN	0.991	0.946	96.35	0.958	0.957	97.08	SCB	0.967	0.9	94.891	0.967	0.967	96.35	
2	ENN	0.938	0.936	94.161	0.973	0.973	97.08	SCB	0.954	0.941	94.891	0.989	0.989	98.54	
3	ENN	0.938	0.945	94.161	0.978	0.977	97.08	SCB	0.956	0.957	95.62	0.968	0.968	97.08	
4	ENN	0.962	0.962	96.296	0.983	0.983	97.778	SCB	0.967	0.946	95.556	0.962	0.962	97.037	
0	EUS	0.943	0.941	94.891	0.958	0.957	96.35	TL	0.933	0.925	92.701	0.983	0.983	97.81	
1	EUS	0.934	0.908	90.511	0.967	0.967	96.35	TL	0.97	0.973	97.08	0.989	0.989	98.54	
2	EUS	0.976	0.973	97.08	0.968	0.968	97.08	TL	0.958	0.957	96.35	0.989	0.989	98.54	
3	EUS	0.963	0.951	94.891	0.978	0.977	97.08	TL	0.929	0.873	91.241	0.921	0.919	94.161	
4	EUS	0.977													

yeast1																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.767	0.765	77.441	0.751	0.751	74.411	IPADE-ID	0.612	0.594	67.34	0.68	0.664	74.074		
1	BC	0.71	0.668	68.687	0.676	0.675	66.667	IPADE-ID	0.663	0.651	68.687	0.627	0.562	74.411		
2	BC	0.723	0.648	60.606	0.74	0.739	72.391	IPADE-ID	0.641	0.588	55.219	0.586	0.456	43.098		
3	BC	0.684	0.645	64.31	0.712	0.711	69.36	IPADE-ID	0.352	0.169	30.976	0.5	0	28.956		
4	BC	0.728	0.706	71.622	0.705	0.7	66.892	IPADE-ID	0.767	0.686	63.176	0.748	0.744	78.041		
0	CNN	0.652	0.647	68.013	0.581	0.426	74.747	NCL	0.757	0.745	74.074	0.618	0.531	75.084		
1	CNN	0.714	0.682	68.013	0.51	0.237	70.034	NCL	0.778	0.715	76.094	0.64	0.564	76.768		
2	CNN	0.723	0.702	70.37	0.529	0.241	72.727	NCL	0.706	0.623	72.727	0.586	0.449	74.411		
3	CNN	0.743	0.736	74.747	0.519	0.215	71.717	NCL	0.711	0.591	75.758	0.599	0.491	74.411		
4	CNN	0.702	0.555	70.608	0.579	0.416	75	NCL	0.772	0.608	74.324	0.596	0.492	73.986		
0	CPM	0.767	0.565	77.104	0.625	0.521	77.104	NM	0.671	0.641	70.034	0.655	0.631	73.064		
1	CPM	0.749	0.73	75.084	0.547	0.305	73.737	NM	0.62	0.577	62.626	0.616	0.607	65.993		
2	CPM	0.726	0.698	74.074	0.557	0.381	72.727	NM	0.681	0.701	70.707	0.682	0.671	73.401		
3	CPM	0.638	0.58	73.401	0.591	0.479	73.737	NM	0.697	0.73	72.391	0.708	0.687	78.114		
4	CPM	0.698	0.532	72.635	0.547	0.339	72.973	NM	0.692	0.644	66.554	0.605	0.569	69.257		
0	ClusterOSS	0.576	0.234	28.62	0.505	0.097	29.63	OSS	0.624	0.533	71.33	0.588	0.45	74.747		
1	ClusterOSS	0.514	0.469	54.882	0.5	0	71.044	OSS	0.708	0.549	71.38	0.548	0.322	73.401		
2	ClusterOSS	0.638	0.566	51.515	0.5	0	28.956	OSS	0.656	0.648	74.074	0.565	0.397	73.401		
3	ClusterOSS	0.618	0.534	61.616	0.617	0.567	72.054	OSS	0.697	0.651	73.737	0.545	0.351	72.054		
4	ClusterOSS	0.574	0.536	50	0.54	0.532	50	OSS	0.766	0.707	80.405	0.607	0.471	77.027		
0	EE	0.743	0.714	68.687	0.726	0.726	71.38	RU	0.649	0.613	57.912	0.729	0.727	70.37		
1	EE	0.757	0.713	69.697	0.715	0.714	70.707	RU	0.763	0.712	72.391	0.679	0.679	66.667		
2	EE	0.719	0.709	74.411	0.679	0.674	64.646	RU	0.777	0.705	65.32	0.733	0.727	69.36		
3	EE	0.772	0.735	72.727	0.766	0.761	73.064	RU	0.723	0.692	73.064	0.715	0.714	70.707		
4	EE	0.715	0.698	72.973	0.663	0.663	65.541	RU	0.686	0.643	69.932	0.681	0.681	65.581		
0	ENN	0.692	0.661	71.717	0.623	0.565	73.401	SCB	0.72	0.699	67.677	0.731	0.724	68.687		
1	ENN	0.717	0.641	75.084	0.667	0.609	78.114	SCB	0.741	0.705	71.38	0.7	0.7	70.034		
2	ENN	0.792	0.737	78.788	0.647	0.591	75.758	SCB	0.68	0.64	62.626	0.666	0.661	63.3		
3	ENN	0.738	0.622	73.737	0.647	0.58	76.768	SCB	0.735	0.707	71.044	0.667	0.657	61.953		
4	ENN	0.717	0.663	74.324	0.628	0.546	76.014	SCB	0.738	0.713	70.608	0.749	0.744	71.284		
0	EUS	0.761	0.696	76.094	0.729	0.724	69.36	TL	0.703	0.674	73.064	0.568	0.41	73.401		
1	EUS	0.68	0.67	66.33	0.705	0.705	70.37	TL	0.68	0.524	67.677	0.543	0.35	71.717		
2	EUS	0.732	0.701	67.677	0.682	0.682	68.013	TL	0.711	0.627	71.38	0.58	0.415	75.084		
3	EUS	0.749	0.722	75.084	0.689	0.689	70.034	TL	0.728	0.627	74.747	0.558	0.369	73.401		
4	EUS	0.724	0.742	75.338	0.736	0.731	69.932	TL	0.733	0.696	78.378	0.599	0.457	76.351		
yeast3																
Fold	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%	Algoritmo	j48auc	j48gm	j48_%	SVMauc	SVMgm	SVM_%		
0	BC	0.922	0.924	91.275	0.911	0.911	88.926	IPADE-ID	0.676	0.425	90.268	0.744	0.708	92.282		
1	BC	0.893	0.892	87.919	0.913	0.913	91.611	IPADE-ID	0.544	0.301	89.597	0.5	0	88.926		
2	BC	0.858	0.888	89.597	0.883	0.882	90.94	IPADE-ID	0.61	0.386	89.262	0.53	0.246	89.597		
3	BC	0.968	0.962	93.289	0.919	0.919	90.268	IPADE-ID	0.75	0.576	92.282	0.513	0.174	88.926		
4	BC	0.888	0.9	89.726	0.882	0.882	89.041	IPADE-ID	0.889	0.874	93.151	0.514	0.179	89.384		
0	CNN	0.857	0.877	92.617	0.674	0.598	91.611	NCL	0.888	0.727	92.282	0.699	0.643	91.275		
1	CNN	0.853	0.86	92.282	0.796	0.779	91.946	NCL	0.865	0.855	94.295	0.725	0.673	93.624		
2	CNN	0.975	0.974	97.651	0.682	0.603	92.953	NCL	0.815	0.801	92.953	0.756	0.716	94.295		
3	CNN	0.947	0.966	96.309	0.648	0.548	91.611	NCL	0.949	0.914	94.295	0.737	0.692	93.289		
4	CNN	0.937	0.934	95.89	0.659	0.567	92.466	NCL	0.933	0.93	95.205	0.803	0.78	95.205		
0	CPM	0.815	0.842	91.611	0.771	0.746	92.282	NM	0.743	0.73	78.523	0.835	0.832	89.597		
1	CPM	0.929	0.941	94.295	0.813	0.793	94.966	NM	0.888	0.843	86.242	0.868	0.866	90.604		
2	CPM	0.938	0.822	93.96	0.813	0.793	94.966	NM	0.67	0.754	77.517	0.922	0.922	93.289		
3	CPM	0.933	0.93	94.631	0.735	0.691	92.953	NM	0.81	0.844	81.879	0.837	0.833	89.933		
4	CPM	0.935	0.932	95.548	0.765	0.733	93.493	NM	0.945	0.937	93.839	0.92	0.92	90.753		
0	ClusterOSS	0.904	0.899	82.886	0.557	0.336	21.141	OSS	0.89	0.907	95.638	0.65	0.549	91.946		
1	ClusterOSS	0.906	0.767	63.423	0.562	0.353	22.148	OSS	0.899	0.893	95.638	0.735	0.691	92.953		
2	ClusterOSS	0.86	0.853	77.517	0.528	0.238	16.107	OSS	0.86	0.841	94.631	0.71	0.65	93.289		
3	ClusterOSS	0.839	0.837	78.523	0.726	0.673	51.342	OSS	0.857	0.835	93.289	0.723	0.672	93.289		
4	ClusterOSS	0.901	0.899	84.932	0.688	0.613	44.178	OSS	0.882	0.866	94.863	0.786	0.759	94.863		
0	EE	0.939	0.947	92.953	0.883	0.882	90.94	RU	0.889	0.907	92.953	0.9	0.9	91.611		
1	EE	0.911	0.932	94.966	0.86	0.86	86.913	RU	0.906	0.907	88.255	0.89	0.89	87.584		
2	EE	0.967	0.94	93.96	0.945	0.945	94.966	RU	0.962	0.907	92.953	0.896	0.896	90.94		
3	EE	0.858	0.9	94.295	0.9	0.9	89.262	RU	0.884	0.888	87.248	0.862	0.862	84.899		
4	EE	0.941	0.91	89.041	0.887	0.886	84.932	RU	0.959	0.957	94.863	0.938	0.937	91.438		
0	ENN	0.902	0.882	93.624	0.843	0.83	95.638	SCB	0.949	0.943	94.631	0.866	0.859	94.966		
1	ENN	0.944	0.905	95.302	0.797	0.774	94.631	SCB	0.926	0.9	94.295	0.723	0.672	93.289		
2	ENN	0.751	0.781	92.282	0.661	0.574	91.611	SCB	0.896	0.758	90.94	0.76	0.729	92.617		
3	ENN	0.873	0.876	95.302	0.811	0.792	94.631	SCB	0.952	0.958	94.966	0.875	0.867	96.644		
4	ENN	0.913	0.875	93.493	0.779	0.753	93.493	SCB	0.886	0.845	93.836	0.754	0.716	94.178		
0	EUS	0.954	0.935	90.94	0.905	0.905	90.268	TL	0.86	0.771	93.96	0.708	0.649	92.953		
1	EUS	0.931	0.94	93.96	0.917	0.917	89.933	TL	0.912	0.904	94.966	0.633	0.52	91.275		
2	EUS	0.908	0.871	84.228	0.894	0.894	90.604	TL	0.966	0.852	90.604	0.786	0.757	94.966		
3	EUS	0.93	0.95	91.275	0.932	0.931	90.268	TL	0.937	0.871	94.295	0.71	0.65	93.289		
4	EUS	0.92	0.882	91.781	0.9	0.9	89.726	TL	0.939	0.947	95.548	0.674	0.593	92.466		
0	IHTS	0.776	0.472	31.208	0.502	0.061	11.409	</								



# Bibliografía

- [1] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020, 9780262018029.
- [2] Néstor Rodríguez Vico. *undersampling en GitHub*. 2018. URL: <https://github.com/NestorRV/undersampling>.
- [3] Guillaume Lemaître, Fernando Nogueira y Christos K. Aridas. “Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning”. En: *Journal of Machine Learning Research* 18.17 (2017), págs. 1-5. URL: <http://jmlr.org/papers/v18/16-365.html>.
- [4] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.
- [5] Corinna Cortes y Vladimir Vapnik. “Support-vector networks”. En: *Machine Learning* 20.3 (1995), págs. 273-297. ISSN: 1573-0565. DOI: 10.1007/BF00994018. URL: <https://doi.org/10.1007/BF00994018>.
- [6] *ARFF wikispace*. 2018. URL: <https://weka.wikispaces.com/ARFF>.
- [7] Mark Hall y col. “The WEKA Data Mining Software: An Update”. En: *SIGKDD Explor. Newsl.* 11.1 (nov. de 2009), págs. 10-18. ISSN: 1931-0145. DOI: 10.1145/1656274.1656278. URL: <http://doi.acm.org/10.1145/1656274.1656278>.
- [8] *WEKA website*. 2018. URL: <https://www.cs.waikato.ac.nz/ml/weka/>.
- [9] Xu-Ying Liu, Jianxin Wu y Zhi-Hua Zhou. “Exploratory Undersampling for Class-imbalance Learning”. En: *Trans. Sys. Man Cyber. Part B* 39.2 (abr. de 2009), págs. 539-550. ISSN: 1083-4419. DOI: 10.1109/TSMCB.2008.2007853. URL: <http://dx.doi.org/10.1109/TSMCB.2008.2007853>.

- [10] Kihoon Yoon y Stephen Kwek. “An Unsupervised Learning Approach to Resolving the Data Imbalanced Issue in Supervised Learning Problems in Functional Genomics”. En: *Proceedings of the Fifth International Conference on Hybrid Intelligent Systems*. HIS '05. Washington, DC, USA: IEEE Computer Society, 2005, págs. 303-308. ISBN: 0-7695-2457-5. DOI: 10.1109/ICHIS.2005.23. URL: <https://doi.org/10.1109/ICHIS.2005.23>.
- [11] Victor H Barella, Eduardo P. Costa y André C. P. L. F. Carvalho. “ClusterOSS: a new undersampling method for imbalanced learning”. En: 2014.
- [12] J. MacQueen. “Some Methods for Classification and Analysis of Multivariate Observations”. En: *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1*. Ed. por L. M. Le Cam y J. Neyman. University of California Press, Berkeley, CA, USA, 1967, págs. 281-297.
- [13] P. Hart. “The Condensed Nearest Neighbor Rule (Corresp.)” En: *IEEE Trans. Inf. Theor.* 14.3 (sep. de 2006), págs. 515-516. ISSN: 0018-9448. DOI: 10.1109/TIT.1968.1054155. URL: <https://doi.org/10.1109/TIT.1968.1054155>.
- [14] Dennis L. Wilson. “Asymptotic Properties of Nearest Neighbor Rules Using Edited Data”. En: 2 (ago. de 1972), págs. 408 -421.
- [15] Salvador García y Francisco Herrera. “Evolutionary Undersampling for Classification with Imbalanced Datasets: Proposals and Taxonomy”. En: *Evol. Comput.* 17.3 (sep. de 2009), págs. 275-306. ISSN: 1063-6560. DOI: 10.1162/evco.2009.17.3.275. URL: <http://dx.doi.org/10.1162/evco.2009.17.3.275>.
- [16] Michael R. Smith, Tony Martinez y Christophe G. Giraud-Carrier. “An Empirical Study of Instance Hardness”. En: 2011.
- [17] Victoria López y col. “Addressing Imbalanced Classification with Instance Generation Techniques: IPADE-ID”. En: *Neurocomput.* 126 (feb. de 2014), págs. 15-28. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2013.01.050. URL: <http://dx.doi.org/10.1016/j.neucom.2013.01.050>.
- [18] Rainer Storn y Kenneth Price. “Differential Evolution &Ndash; A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. En: *J. of Global Optimization* 11.4 (dic. de 1997), págs. 341-359. ISSN: 0925-5001. DOI: 10.1023/A:1008202821328. URL: <https://doi.org/10.1023/A:1008202821328>.
- [19] J. Zhang e I. Mani. “KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction”. En: *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets*. 2003.

- [20] Jorma Laurikkala. “Improving Identification of Difficult Small Classes by Balancing Class Distribution”. En: *Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine. AIME ’01.* London, UK, UK: Springer-Verlag, 2001, págs. 63-66. ISBN: 3-540-42294-3. URL: <http://dl.acm.org/citation.cfm?id=648155.757340>.
- [21] Miroslav Kubat y Stan Matwin. “Addressing the Curse of Imbalanced Training Sets: One-Sided Selection”. En: *In Proceedings of the Fourteenth International Conference on Machine Learning.* Morgan Kaufmann, 1997, págs. 179-186.
- [22] Ivan Tomek. “Two modifications of CNN”. En: 6 (nov. de 1976).
- [23] Show-Jane Yen y Yue-Shi Lee. “Under-Sampling Approaches for Improving Prediction of the Minority Class in an Imbalanced Dataset”. En: *Intelligent Control and Automation: International Conference on Intelligent Computing, ICIC 2006 Kunming, China, August 16–19, 2006.* Ed. por De-Shuang Huang, Kang Li y George William Irwin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, págs. 731-740. ISBN: 978-3-540-37256-1. DOI: 10.1007/978-3-540-37256-1\_89. URL: [https://doi.org/10.1007/978-3-540-37256-1\\_89](https://doi.org/10.1007/978-3-540-37256-1_89).
- [24] *The Scala Programming Language.* 2018. URL: <https://www.scala-lang.org>.
- [25] *sbt: Simple Build Tool.* 2018. URL: <https://www.scala-sbt.org/index.html>.
- [26] Frederic P. Miller, Agnes F. Vandome y John McBrewster. *Apache Maven.* Alpha Press, 2010. ISBN: 6130652194, 9786130652197.
- [27] *Scala: Case Classes.* 2018. URL: <https://docs.scala-lang.org/tour/case-classes.html>.
- [28] *Scaladoc.* 2018. URL: <https://docs.scala-lang.org/style/scaladoc.html>.
- [29] *GitHub Pages.* 2018. URL: <https://pages.github.com>.
- [30] *Traits en Scala.* 2018. URL: <https://docs.scala-lang.org/tour/traits.html>.
- [31] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files.* RFC 4180. IETF, oct. de 2005.
- [32] J. Alcalá-Fdez y col. “KEEL: a software tool to assess evolutionary algorithms for data mining problems”. En: *Soft Computing* 13.3 (2009), págs. 307-318. ISSN: 1433-7479. DOI: 10.1007/s00500-008-0323-y. URL: <https://doi.org/10.1007/s00500-008-0323-y>.
- [33] *KEEL website.* 2018. URL: <http://keel.es>.

- [34] Tom Fawcett. “An introduction to ROC analysis”. En: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, págs. 861 -874. ISSN: 0167-8655. DOI: <https://doi.org/10.1016/j.patrec.2005.10.010>. URL: <http://www.sciencedirect.com/science/article/pii/S016786550500303X>.
- [35] *Python Software Foundation*. 2018. URL: <https://www.python.org>.
- [36] F. Pedregosa y col. “Scikit-learn: Machine Learning in Python”. En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.
- [37] Ricardo Barandela y col. “Strategies for Learning in Class Imbalance Problems”. En: 36 (mar. de 2003), págs. 849-851.
- [38] *Banana dataset - KEEL*. 2018. URL: <http://sci2s.ugr.es/keel/dataset.php?cod=182>.
- [39] J. D. Hunter. “Matplotlib: A 2D graphics environment”. En: *Computing In Science & Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.
- [40] Matei Zaharia y col. “Apache Spark: A Unified Engine for Big Data Processing”. En: *Commun. ACM* 59.11 (oct. de 2016), págs. 56-65. ISSN: 0001-0782. DOI: 10.1145/2934664. URL: <http://doi.acm.org/10.1145/2934664>.
- [41] *Apache Spark website*. 2018. URL: <https://spark.apache.org>.

