

Universidades de Burgos, León y
Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros



**TFM del Máster Inteligencia de Negocio
y Big Data en Entornos Seguros**

**Métodos de clasificación
semisupervisados con Spark-ML.
Estudio e Implementación**

Presentado por David Guinart Platero
en Universidad de Burgos — 25 de junio
de 2021

Tutores: Álvar Arnaiz González y Juan José
Rodríguez Diez

Universidades de Burgos, León y Valladolid



Máster universitario en Inteligencia de Negocio y Big Data en Entornos Seguros

Álvar Arnaiz González y Juan José Rodríguez Diez profesor del departamento de Ingeniería informática, área de Lenguajes y Sistemas Informáticos.

Expone: Que el alumno D. David Guinart Platero, con DNI 47648288Q, ha realizado el Trabajo final de Máster en Inteligencia de Negocio y Big Data en Entornos Seguros: *Métodos de clasificación semi-supervisados con Spark-ML. Estudio e Implementación.*

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 25 de junio de 2021

Vº. Bº. del Tutor:

D. Álvar Arnaiz González

Vº. Bº. del co-tutor:

D. Juan José Rodríguez Diez

Resumen

En este presente trabajo se van a estudiar diferentes algoritmos de clasificación en el entorno de aprendizaje semisupervisado (inductivo) trabajando con clasificadores base (Decision tree, Naive Bayes...) utilizando Spark ML. El objetivo del mismo es hacer su implementación, su comprobación y la creación de una biblioteca con Spark ML para facilitar su uso.

Por otro lado, también se hace un estudio (experimentación) con diferentes conjuntos de datos donde la idea principal del mismo es hacer una comparativa empírica entre los resultados conseguidos a partir de los clasificadores base trabajando de forma supervisada (solo teniendo en cuenta el conjunto de datos etiquetados) con diferentes algoritmos de aprendizaje semi-supervisado (teniendo en cuenta los datos etiquetados y no etiquetados). Para ello, se ha creado un dashboard para su fácil interpretación

Finalmente, se expone sus conclusiones juntamente con sus futuras líneas de trabajo que este proyecto no ha podido cubrir con el objetivo de utilizar la biblioteca generada antes mencionada.

Descriptores

Semisupervisado, supervisado, algoritmos, Self-Training, Co-Training, machine learning, Spark, ML, Databricks, Python, Scala, Pipelines...

Abstract

This master thesis studies different classification algorithms within the frame of semi-supervised learning (inductive) working with base classifiers (Decision tree, Naive Bayes...) using Spark ML.

The main goal of this study is to design, implement, verify and build a library on Spark ML in order to make it easy to reuse on the any Spark environment. On the other hand, this project do an experimentation with different datasets where the main idea is to compare empirically the results between supervised algorithms /base classifiers (working with the samples labeled) and the semi-supervised algorithms working with semi-supervised (samples labeled and unlabeled). In order to analyze in more details, the outcomes and manage the huge number of data, this project has created a dashboard.

Finally, this project presents the conclusions and new lines of research working with the new library built (for Spark) in this research project as it has been described previously.

Keywords

Semi-supervised, supervised, algorithms, Self-Training, Co-Training, machine learning, Spark, ML, Databricks, Python, Scala, Pipelines...

Índice general

Índice general	IV
Índice de figuras	VII
Índice de tablas	XI
Memoria	1
1 Introducción	3
2 Objetivos del proyecto	5
2.1. Objetivos generales	5
2.2. Objetivos técnicos	5
2.3. Objetivos personales	5
3 Conceptos teóricos	7
3.1. Introducción	7
3.2. Métodos SSC	9
3.3. SSC Wrapper methods - Self Labeled	11
4 Técnicas y herramientas	15
4.1. Spark	15
4.2. Spark ML/MLlib	18
4.3. DataBricks	20
4.4. KEEL	22
4.5. Power BI	23
4.6. Python SeaBorn y Matplotlib	24

5 Aspectos relevantes del desarrollo del proyecto	25
5.1. Datos utilizados	26
5.2. Algoritmos supervisados - Clasificadores base	26
5.3. Algoritmos semi-supervisados implementados	27
5.4. Verificación y test de los algoritmos SSC implementados . . .	29
6 Trabajos relacionados	39
7 Conclusiones y Líneas de trabajo futuras	45
Apéndices	47
Apéndice A Plan de Proyecto	49
A.1. Introducción	49
A.2. Planificación temporal	50
Apéndice B Experimentación	53
B.1. Introducción	53
B.2. Objetivos generales	53
B.3. Resultados con los clasificadores de aprendizaje supervisado	54
B.4. Relación semisupervisado el threshold y el porcentaje de etiquetado seleccionado	56
B.5. Comparativa semisupervisado vs supervisado	61
B.6. Comparativa Self-Training vs Co-Training	66
B.7. Conclusiones de los experimentos	70
B.8. Relación entre los datos Etiquetados (inicio del y final) . . .	71
Apéndice C Especificación de diseño	73
C.1. Introducción	73
C.2. Diseño en Spark	73
C.3. Diseño arquitectónico	74
Apéndice D Documentación técnica de programación	77
D.1. Introducción	77
D.2. Versiones utilizadas para DataBricks, Spark, Scala, Python y Power BI	77
D.3. Clases y Objetos	78
D.4. Creación de las librerías para los algoritmos semisupervisados	79
D.5. Como añadir el paquete .jar en DataBricks	81
Apéndice E Documentación de usuario	85

E.1. Introducción	85
E.2. Utilización de las librerías <i>org.apache.spark.ml.semisupervised._</i>	85
Bibliografía	91

Índice de figuras

3.1. Tipos de aprendizaje en Machine Learning [17]	8
3.2. Número de publicaciones por año en el campo del aprendizaje semisupervisado [40]	9
3.3. Taxonomía SSC visión general	9
3.4. Taxonomía Wrapper methods	11
3.5. Concepto Self-Labeled	12
4.1. Estructura componentes Spark	17
4.2. Arquitectura Spark (alto nivel)[1]	18
4.3. Métricas en Spark MLlib [12]	19
4.4. Workflow y pipeline	20
4.5. Configuración del cluster de ejecución en DataBricks	22
4.6. Métodos implementados en Keel [7]	23
5.1. Ilustración funcionamiento algoritmo Self-Training.	28
5.2. Ilustración funcionamiento algoritmo Co-Training.	31
5.3. Resultados clasificadores base (supervisados) desetiquetando instancias	32
5.4. Resultados Keel vs Spark Decision Tree (DT) supervisado . . .	33
5.5. Resultados Keel vs Spark, Supervisado Decision Tree/ árbol de decisión (DT) como clasificador base, con todos los datos etiquetados	34
5.6. Resultados Keel vs Spark, Self-Training (ST) con DT como clasificador base ST (DT)	34
5.7. Resultados Keel vs Spark, Self-Training con DT como clasificador base, cada gráfico representa un porcentaje de etiquetado . . .	35
5.8. Resultados Keel vs Spark, Self-Training (CT) con CT como clasificador base ST(DT)	36

5.9. Resultados Keel vs Spark, Co-Training con DT como clasificador base, cada gráfico representa un porcentaje de etiquetado	37
6.1. Artículo [26] accuracy Self-Training y Co-Training	40
6.2. Artículo [26] Self-Training y Co-Training. Accuracy vs datos etiquetados y no etiquetados	40
6.3. Figura extraída del artículo [31] Comportamiento con diferentes distribuciones de clase	41
6.4. Figura extraída del artículo [31] Comportamiento con diferentes tamaños en los conjuntos de validación	42
6.5. Figura extraída del artículo [31] Comportamiento con diferentes tamaños entre datos etiquetados y no etiquetados	42
A.1. Roadmap, planificación del proyecto	50
B.1. Comparativa con diferentes métricas clasificadores base (supervisados). El resultado es el valor medio con todos los conjuntos de datos.	55
B.2. Comparativa con diferentes métricas clasificadores base (supervisados). Resultado del conjunto de datos coil2000	55
B.3. Valor medio accuracy con todos los conjuntos de datos menores a 20k. Resultado entre threshold y porcentaje etiquetado Self-Training	57
B.4. Valor medio accuracy con todos los conjuntos de datos menores a 20k de instancias con Random Forest (RF). Resultado entre threshold y porcentaje etiquetado Self-Training	57
B.5. Valor medio accuracy con todos los conjuntos de datos menores a 20k de instancias. Resultado entre el porcentaje etiquetado y el threshold Self-Training	58
B.6. Valor medio accuracy con todo los conjunto de datos mayores a 1M. Resultado entre threshold y porcentaje etiquetado Self-Training	58
B.7. Valor medio accuracy con todos los conjuntos de datos menores a 1M de instancias. Resultado entre el porcentaje etiquetado y el threshold Self-Training	59
B.8. Valor medio accuracy con todo los conjunto de datos Mayores a 1M de instancias con Random Forest (RF). Resultado entre threshold y porcentaje etiquetado Self-Training	59
B.9. Valor medio accuracy con todo los conjunto de datos menores a 20k de instancias. Resultado entre el porcentaje etiquetado y el threshold Co-Training	60

B.10. Valor medio accuracy con todo los conjunto de datos mayores a 1M de instancias. Resultado entre el porcentaje etiquetado y el threshold Co-Training	61
B.11. Valor medio accuracy y porcentaje etiquetado para Self-Training y clasificadores Supervisado con todo los conjunto de datos menores a 20k instancias. Comparativa entre Self-Training vs Supervisado	62
B.12. Accuracy y porcentaje etiquetado para Self-Training y RF con el conjunto de datos heart. Comparativa entre Self-Training vs Supervisado	62
B.13. Accuracy y porcentaje etiquetado para Self-Training y RF con el conjunto de datos sonar. Comparativa entre Self-Training vs Supervisado	63
B.14. Valor medio accuracy y porcentaje etiquetado para Self-Training y clasificadores Supervisado con todo los conjunto de datos mayores a 1M de instancias. Comparativa entre Self-Training vs Supervisado	64
B.15. Valor medio accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con todo los conjunto de datos menores a 20k de instancias. Comparativa entre Self-Training vs Supervisado	65
B.16. Accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con el conjunto heart. Comparativa entre Self-Training vs Supervisado	65
B.17. Accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con el conjunto sonar. Comparativa entre Self-Training vs Supervisado	66
B.18. Valor medio accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con todo los conjunto de datos mayores a 1M de instancias. Comparativa entre Self-Training vs Supervisado	67
B.19. Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con todo los conjunto de datos menores a 20k instancias. Comparativa entre Self-Training vs Co-Training	68
B.20. Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training el conjunto de datos sonar con RF. Comparativa entre Self-Training vs Co-Training	68
B.21. Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con todos los conjunto de datos menores a 20k instancias. Comparativa entre Self-Training vs Co-Training	69
B.22. Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con el conjunto de datos Poker. Comparativa entre Self-Training vs Co-Training	69

B.23. Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con el conjunto de datos TXNY. Comparativa entre Self-Training vs Co-Training	70
B.24. Relación entre el etiquetado inicial, final y el porcentaje etiquetado	71
C.1. Pipeline utilizado para el diseño del proyecto	74
C.2. Arquitectura y diseño utilizado	75
D.1. Creación del paquete .jar	82
D.2. Añadir paquete .jar paso-1	83
D.3. Añadir paquete .jar paso-2	83
D.4. Añadir paquete .jar paso-3	84
D.5. Añadir paquete .jar paso-4	84
E.1. Importamos las librerías semisupervised.	86
E.2. Leemos del conjunto de datos y realizamos el proceso de featurization.	87
E.3. División en datos de entrenamiento y test.	87
E.4. Desetiquetamos las clases.	87
E.5. Entrenamos el modelo para Self-Training con RF y vemos los resultados obtenidos con este modelo.	88
E.6. Obtenemos los resultados de los datos etiquetado y no etiquetados al inicio y final del proceso de Self-Training.	88
E.7. Entrenamos el modelo para Co-Training con RF y vemos los resultados obtenidos con este modelo.	89

Índice de tablas

3.1. SSC - métodos	14
5.1. Algoritmos semisupervisados con mejores resultados según el artículo [32].	25
5.2. Algoritmos semisupervisados y clasificadores base en Spark seleccionados.	26
5.3. Datos utilizados en el diseño y en los experimentos (*después del pre-procesado)	26
5.4. Clasificadores base métodos	27
D.1. definición de la clase SemiSupervisedDataResults	78
D.2. definición de la clase UnlabeledTransformer	78
D.3. definición de la clase Supervised	79
D.4. Definición de la clase SeftTraining	80
D.5. Definición de la clase Co-Training	81
D.6. Definición del objeto functionsSemiSupervised	82

Memoria

Capítulo 1

Introducción

En este proyecto se van a estudiar diferentes algoritmos de clasificación en la tarea de aprendizaje semisupervisado (inductivo) trabajando con clasificadores base utilizando Spark ML. El objetivo del mismo es realizar su diseño, implementación, su comprobación y la creación de una biblioteca con Spark ML para facilitar su uso simplemente importando la misma en cualquier entorno Spark. Los métodos semisupervisados implementados serán concretamente: Self-Training y Co-Training.

Esta implementación se realizará con Spark utilizando la librería de ML (evolución de la librería previa de Machine Learning MLlib), dado su flexibilidad, escalabilidad... lo hacen un candidato muy idóneo para trabajar con grandes conjunto de datos (Big Data). Descrito en el capítulo 4.

Posteriormente a su diseño e implementación, se ha empaquetado en una biblioteca para poder ser importada sin ningún tipo de implementación extra trabajando con Spark, la descripción de las mismas están explicadas en los apéndices D y E. Esto servirá para posibles trabajos posteriores sobre estos algoritmos o para la implementación de nuevos algoritmos semisupervisados en el futuro, todas las implementaciones están publicadas de forma abierta en Github <https://github.com/Dguipla/TFM-SemiSup>.

Finalmente, se han realizado varios experimentos tanto para demostrar su correcto funcionamiento (Sección 5.4), como una comparativa con otros algoritmos supervisados, demostrando que los algoritmos semisupervisados pueden tener mejor rendimiento que los supervisados para diferentes casuísticas y diferentes conjunto de datos (apéndice B).

Este trabajo se estructura como sigue:

En el capítulo 3.1 se explican los conceptos teóricos sobre los algoritmos semisupervisados y sus principales ventajas. Seguidamente en el capítulo 4 se presentan las diferentes tecnologías utilizadas tanto para el diseño e implementación de los algoritmos (que para este caso es Spark con DataBricks) como para su visualización donde se utiliza Python y Power BI.

En el capítulo 5 se presentan los algoritmos Self-Training y Co-Training en más detalle con sus pseudocódigos, también se presentan los datos que se van a utilizar para los experimentos que se exponen en los apéndices y por último se hace una comparativa de los algoritmos diseñados en Spark con la herramienta de KEEL para validar si los métodos (Self-Training y Co-Training) han sido bien implementados.

En el capítulo 6 se hace una referencia a diferentes trabajos que están relacionados con los tópicos de este proyecto. Por último, en el capítulo 7 se exponen las conclusiones así como futuras líneas de trabajo.

En relación a los apéndices (no menos importantes) se organizan como siguen:

El apéndice A hace referencia al plan del proyecto (*roadmap*) y las herramientas utilizadas para gestionar el mismo.

El apéndice B presenta los resultados con los conjuntos de datos presentados en el capítulo 5 y su comparativa entre supervisado vs semisupervisado.

A continuación en el apéndice C se explica el diseño utilizado y el pipeline implementado con Spark ML así como la arquitectura utilizada.

Finalmente en los apéndices D y E se describen las clases, la creación de librerías para Spark, ejemplo de utilización de las librerías Self-Training y Co-Training creadas entre otros conceptos relevantes para el usuario o programador.

Capítulo 2

Objetivos del proyecto

2.1. Objetivos generales

En este presente trabajo se van a estudiar diferentes algoritmos de clasificación en el entorno de aprendizaje semisupervisado (inductivo) trabajando con clasificadores base utilizando Spark ML.

La idea principal del mismo es hacer una comparativa empírica entre los resultados conseguidos a partir de los clasificadores base trabajando de forma supervisada (solo teniendo en cuenta el conjunto de datos etiquetados) con diferentes algoritmos de aprendizaje semisupervisado (teniendo en cuenta los datos etiquetados y no etiquetados).

2.2. Objetivos técnicos

Desde el punto de vista técnico se va a utilizar Spark ML y sus clasificadores supervisados para el diseño de los algoritmos semisupervisados, el principal motivo es su versatilidad, escalabilidad y su procesamiento distribuido que hace que sea una herramienta ideal para trabajar con un gran volumen de datos.

2.3. Objetivos personales

Por último desde el punto de vista personal, el principal objetivo es profundizar en los conocimientos sobre Spark y Scala así como en la introducción del aprendizaje semisupervisado el cual tiene una demanda significativa en los últimos años como se ha descrito en la memoria.

Capítulo 3

Conceptos teóricos

3.1. Introducción

En la actualidad se pueden distinguir dos grandes tipos de aprendizaje automático, principalmente estos son aprendizaje supervisado, donde contamos con clases etiquetadas es decir con un conocimiento a priori, y por el otro lado el aprendizaje no supervisado, donde no se tienen instancias etiquetadas y no hay un conocimiento previo. En este trabajo vamos a introducir otro tipo de aprendizaje dentro del machine learning concretamente el aprendizaje semisupervisado (Semi Supervised Learning–SSL), Figura 3.1.

La característica principal de este es que utiliza un conjunto de datos etiquetados (L) y conjunto de datos no etiquetados U para el entrenamiento y la creación del modelo. Con lo cual utilizando el conjunto L se generará un modelo que etiquetará en gran medida parte del conjunto U para posteriormente tener un conjunto de datos mayor y generar el modelo final.

L es el conjunto de datos etiquetados para el entrenamiento [25]:

$$L = (x_i, y_i) \quad x_i \in \mathbb{R}^d, y_i \in \Omega, i = 1, \dots, l$$

Donde cada instancia es descrita por un vector d -dimensional de atributos (features) $x_i \in \mathbb{R}^d$ e y_i es la clase etiquetada para cada vector x_i . Por otro lado $\Omega = \omega_1, \dots, \omega_K$ es el conjunto de diferentes clases de etiquetas estas pueden ser binaria dos tipos de clases o múltiples (más de dos tipos de clases). Por otro lado [25]:

$$U = x_j^* | j = 1, \dots, u$$

Uno de los motivos principales por el cual se utiliza el aprendizaje semisupervisado en Machine Learning es que el proceso de etiquetado en la mayoría de ocasiones es largo y tedioso donde se convierte en una tarea repetitiva expuesta posibles errores. Con lo cual hay que tener en cuenta que normalmente el tamaño entre los dos conjunto de datos que se utilizarán para entrenar y generar el modelo tanto L como U sus tamaños van a ser muy diferentes principalmente el porcentaje de datos etiquetados va a ser muy menor al de los no etiquetados por los motivos antes mencionados $l \ll u$.

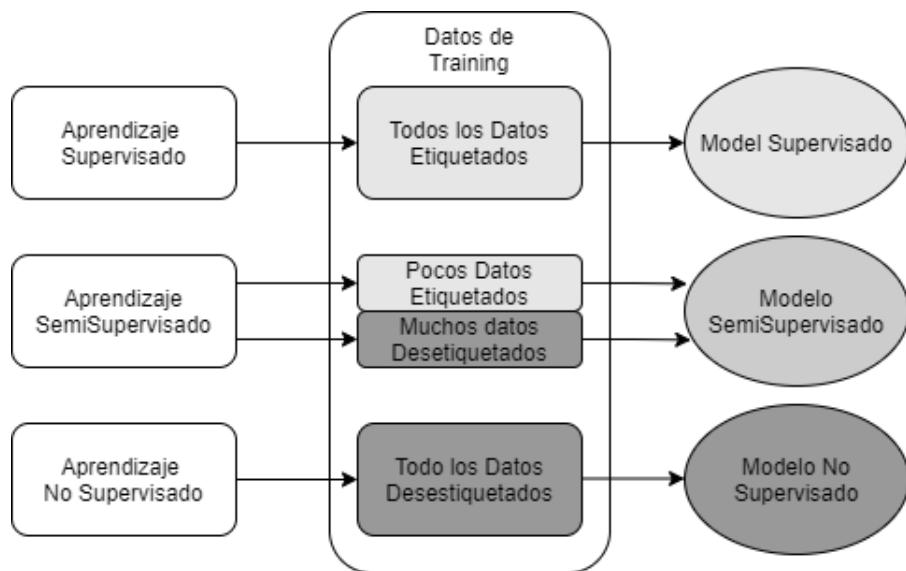


Figura 3.1: Tipos de aprendizaje en Machine Learning [17]

Cabe remarcar que en la actualidad el aprendizaje semisupervisado es un tipo de aprendizaje que está en alza. Como se puede ver en la Figura 3.2 el número de publicaciones está en constante crecimiento en los últimos diez años. Con lo cual se podría afirmar que es una técnica en auge, utilizada actualmente y que pude ser de gran utilidad para determinados casos de uso.

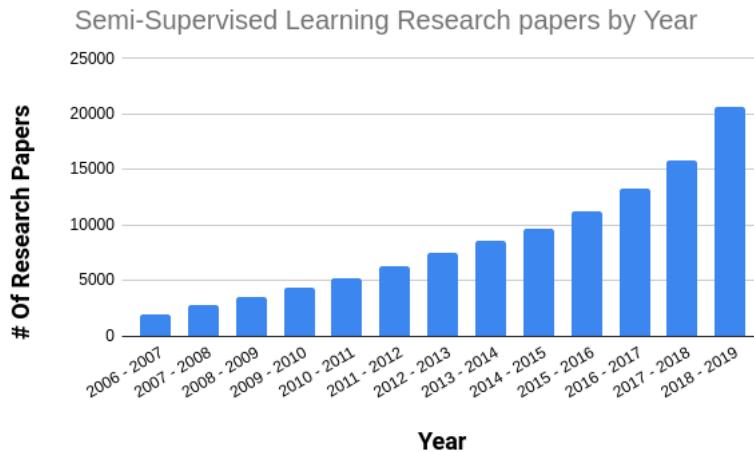


Figura 3.2: Número de publicaciones por año en el campo del aprendizaje semisupervisado [40]

3.2. Métodos SSC

En este apartado se pretende dar una visión general de los diferentes algoritmos dentro del SSC (clasificación semi-supervisada).

SSC se puede dividir en 2 grandes grupos aprendizaje: transductivo e inductivo [32]

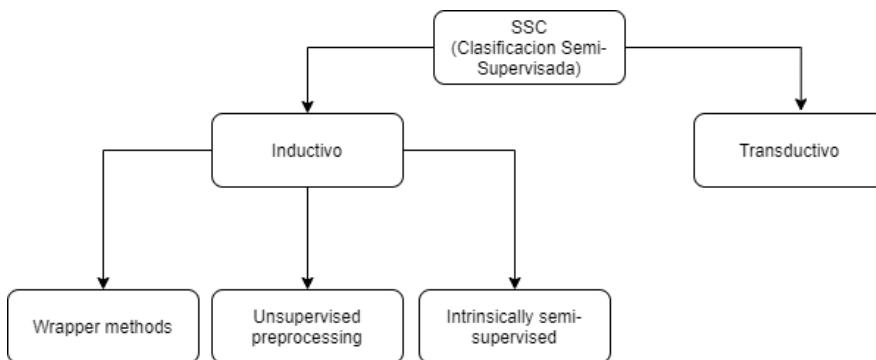


Figura 3.3: Taxonomía SSC visión general

1. **Transductivo:** No genera ningún modelo con los datos de entrenamiento, en este caso todos los datos trabajan conjuntamente (tanto los de entrenamiento como los de test no hay distinción entre ellos) y

se utilizan para crear el aprendizaje y generar las predicciones finales, algunas de sus características:

- No se genera un modelo que posteriormente se reutiliza para diferentes conjuntos de datos de test solo se etiquetarán los datos que han sido utilizados para el aprendizaje.
- Cuando se introducen nuevos datos que se desean etiquetar, se tiene que generar nuevamente la predicción (proceso de aprendizaje) para este nuevo conjunto trabajando juntamente con los datos de entrenamiento.
- Si hay pocos datos de entrenamiento puede tener mejor rendimiento que el caso inductivo ya que podrían clasificar mejor los datos de test si trabajan juntos con los de entrenamiento para su predicción
- Más recursos de computación.

2. **Inductivo:** Este caso es el más convencional dentro del entorno ML, es decir utiliza los datos de entrenamiento para entrenar un modelo donde posteriormente una vez esté generado servirá para predecir las etiquetas de los datos de test. Con lo cual, se puede reutilizar el modelo con diferentes conjuntos de test sin necesidad de re-entrenar el modelo. Dentro de los métodos inductivos se suelen distinguir tres grupos (Figura 3.5) [33]:

- a) **Wrapper methods:** También llamado **Self-Labeled**. La principal idea detrás de este método es que en él hay siempre uno o varios modelos de aprendizaje supervisado los cuales van a ser utilizados para entrenar el conjunto de datos etiquetado y posteriormente este va a ser utilizado para etiquetar el conjunto de dato sin etiquetas. En comparación con otros métodos este puede usar casi cualquier método de aprendizaje supervisado existente como subrutina, con lo cual ayuda al desarrollo sin derivar en la necesidad de hacer nuevos algoritmos desde cero. Remarcar que en este trabajo se va profundizar en esta metodología en los próximos capítulos.
- b) **Unsupervised processing:** A diferencia del caso anterior, el Unsupervised processing utiliza los datos etiquetados y no etiquetados en dos etapas separadas, donde va a poder extraer información de los datos no etiquetados trabajando de forma no supervisada (conociendo las etiquetas de los datos etiquetados)

pero sin trabajar con ellos) etiquetando los mismos a partir de agrupación o clustering.

- c) **Intrinsically semisupervised:** Estos tipos de métodos no se basan en pasos intermedios o en métodos base de aprendizaje supervisado, son capaces de realizar el aprendizaje utilizando ambos conjuntos de datos (etiquetado y no etiquetado). Para poner un ejemplo tendríamos el método (S3VMs) [30] donde no utiliza ningún clasificador base o método base que provenga de un aprendizaje supervisado ya conocido como por ejemplo clasificador Naïve Bayes, en si mismo el método trabaja con ambos tipos de datos.

3.3. SSC Wrapper methods - Self Labeled

Como se ha descrito previamente hay varios grupos y subgrupos de métodos para SSC. Este apartado se va a centrar en los métodos Self Labeled (Figura 3.4) [32], dado que va a ser el utilizado para este trabajo (Secciones 5, C, B) donde se van a describir sus características y algunos de sus métodos juntamente con algunos de sus posibles clasificadores base (aprendizaje supervisado).

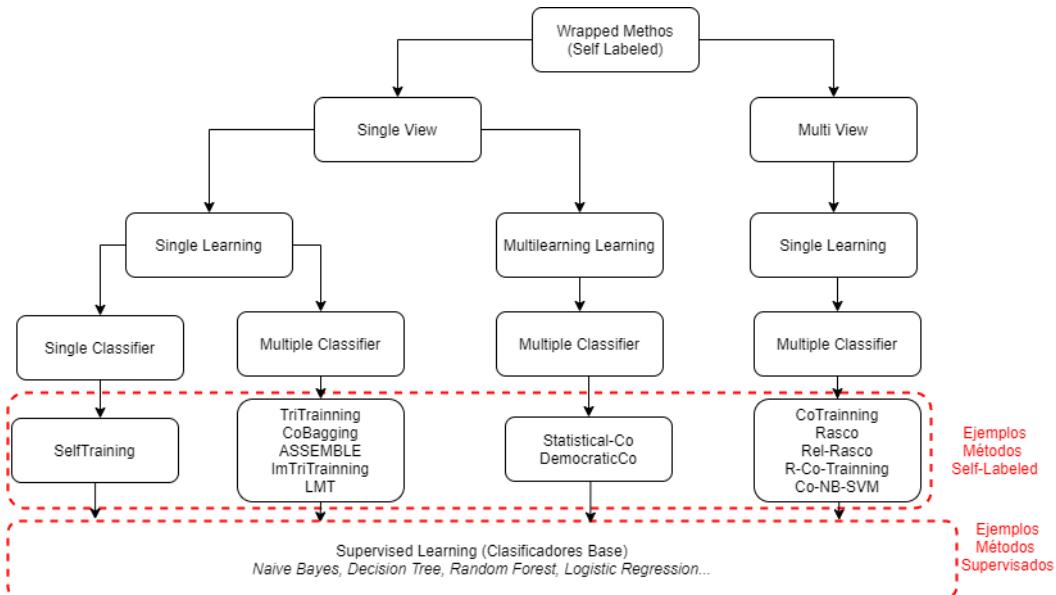


Figura 3.4: Taxonomía Wrapper methods

Los wrapper method/self-labeled (ver sección 3.2) trabajan con clasificadores de aprendizaje supervisado ya previamente conocidos para etiquetar los datos de entrenamiento no etiquetados, con lo cual la idea principal es utilizar los datos etiquetados L y los no etiquetados U como conjuntos de entrenamiento, donde¹:

1. **Entrena el modelo** con los datos etiquetados L . Se tiene que tener en cuenta que en muchas ocasiones el etiquetado se hace manualmente como se comentó en el apartado 3.1.
2. **Utilización del modelo** para etiquetar los datos del conjunto de datos no etiquetados U .
3. **Predicción** del conjunto U
4. **Agrupamiento**² dentro los dos conjuntos $L \cup U$
5. Generación del **modelo final** con los datos agrupados en el punto anterior

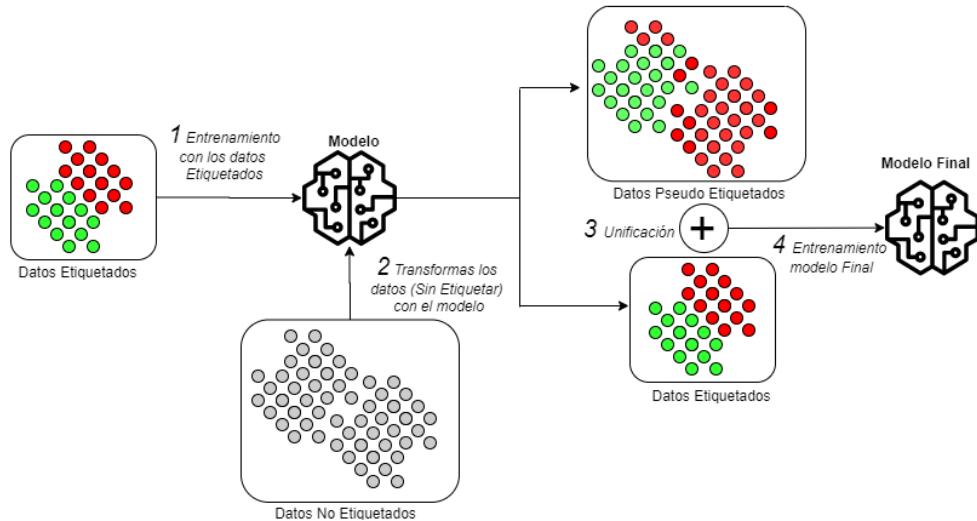


Figura 3.5: Concepto Self-Labeled

¹Es una aproximación (visión general del concepto SSC) en ningún caso es un método final SSC estos los veremos a continuación(ver Figura 3.5)

²En función de la probabilidad de predicción se incluirán o no en el nuevo conjunto de datos

Propiedades Wrapper methods-Self Labeled

Visto un resumen general (Figura 3.5) de la metodología self-labeled, en este apartado se va a adentrar en las características/propiedades que tienen los diferentes métodos SSC (Self-Labeled) [32].

- **Single-view y Multi-view:** La idea principal de esta propiedad es la selección de atributos (features) y de los datos para el conjunto de datos etiquetados L . Los métodos Self-Labeled que utilizan Multi-view, L es dividida en 2 o más subsets de atributos (L_{s_k}) de dimensiones M donde $M < D$. Multi-view necesita datos redundantes e independientes siempre y cuando se proporcione las suficientes ejemplos en cada subset (L_{s_k}) para que se pueda entrenar correctamente, por todo, multi-view necesita trabajar con múltiples clasificadores. Por otro lado, single-view utilizará el conjunto de datos L en su totalidad sin hacer ninguna división de ellos. Remarcar que single-view puede utilizar un solo clasificador o múltiples clasificadores con la totalidad de su conjunto de datos etiquetados L .
- **Single-learning y Multi-learning:** La diferencia entre los dos es que multi-learning hace una combinación entre diferentes tipos de algoritmos de aprendizaje SSC independientemente del número de clasificadores base que pueda utilizar [22]. Es decir, en el caso de utilizar múltiples clasificadores base, pero solo un algoritmo de aprendizaje en este caso tendríamos la propiedad single-learning en nuestro método SSC.
- **Single-classifier y Multiple-classifier:** Como se ha comentado para el proceso de etiquetaje de los datos no etiquetados del conjunto U y su posterior $L \cup U$ para ello se necesitan clasificadores base supervisados. La diferencia entre single y multiple-classifier se base en el número de clasificadores base (supervisados) que el método va utilizar, como modo de ejemplo para el caso de Self-Trainning [36] este utiliza un clasificador base en su método, por otro lado Co-Traininig [20] y TriTraining [38] utilizan dos y tres respectivamente. La principal idea es obtener una mejora del rendimiento con más de un clasificador y un conjunto reducido de instancias que con un único clasificador. El inconveniente en esto es que la implementación de estos métodos es más compleja y los recursos/tiempo necesario se incrementan considerablemente.
- **Métodos SSC:** Según la Figura 3.4 hay diferentes tipos de métodos dependiendo de sus características (definidas previamente) en este

punto se van a presentar algunos de ellos y en que artículos han sido diseñados, por otro lado en la fase experimental y de diseño (secciones/apéndices B, 5, C) se verá la implementación y la comparación entre estos y los clasificadores base utilizando Spark ML y KEEL [32].

Método	Nombre completo	Referencia
Self-Training	Standard SelfTrainig	[36]
ASSEMBLE	ASSEMBLE	[19]
ImTriTraining	Improved Tri-training	[24]
TriTraining	TriTraining	[39]
Statistical-Co	Statistical-Colearing	[23]
Co-Training	Standard Co-Training	[20]
CoForest	CoForest	[29] , [26]
Rasco	Random subspace method for co-training	[35]
R-Co-Training	Robust co-training	[27]
Co-NB-SVM	Co-training with NB and SVM classifiers	[28]
DemocraticCo	Democratic co-learning	[37]

Tabla 3.1: SSC - métodos

- **Clasificadores Base:** Por último mucho de los métodos antes mencionados utilizan clasificadores utilizados en aprendizaje supervisado como base de sus algoritmos con el objetivo de aprender de los datos etiquetados (L) al inicio y posteriormente de estos mismos más lo que se van etiquetando de los datos no etiquetados U durante el proceso.

- Árbol de decisión
- Naïve Bayes
- Regresión logística
- KNN
- Random Forest
- SVC
- Otros

En este trabajo, para la parte experimental se va a centrar en clasificadores probabilísticos existentes en Spark ML.

Capítulo 4

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto.

Con lo cual se va hacer referencia por un lado a las herramientas utilizadas haciendo una presentación de las mismas y poniéndolas en contexto (ver sección 4.1) como sus características principales y las técnicas utilizadas (ver sección 4.2).

4.1. Spark

En los capítulos previos se han explicado los métodos SSC así como sus características principales. Para ello en este trabajo se pretende utilizar Spark y su librería ML como clasificadores base (supervisados) e implementar diferentes métodos SSC en Spark. En este apartado vamos a hacer una introducción general sobre Spark y sus aspectos más destacados.

Spark como definición general es una plataforma de código abierto (open-source), es decir es una plataforma gratuita para procesado distribuido, rápido de datos a gran escala y de propósito general. Spark esta basado en computación en memoria en el cluster con lo cual tiene un mejor rendimiento que otros frameworks anteriores como Hadoop.

Criterios de selección de Spark

En términos de rendimiento Spark es hasta 100 veces mas rápido si se ejecuta en memoria que Apache Hadoop y 10 veces más rápido si este lo

hace en disco. Por otro lado también aquí enumeramos una serie de ventajas como:

- Fácil adaptación ya que es una solución open-source.
- Escalable y adaptable en función del caso de uso a realizar, con lo que se pueden añadir mas nodos al cluster sin tener que modificar el diseño.
- APIs para Java, Python, Scala y R principalmente.
- Integración con las bases de datos no relacionales (NoSQL) más destacadas como:
 - Cassandra
 - HBase
 - MongoDB
 - *Otras*
- Tolerancia a fallos
- Librerías de alto nivel como por ejemplo librerías de ML (como posteriormente describiremos), librerías de stream (flujos) analytics entre otras.

Componentes Spark

Los componentes de Spark se distribuyen como sigue (Figura 4.1):

- **Spark Core** Es el núcleo donde se apoya toda la arquitectura de Spark, donde incluye componentes para la distribución, planificación y monitorización. Contiene el modelo que permite a las operaciones como map, reduce, filter... operar en paralelo sobre RDD. Por encima de este tendríamos 4 módulos:
 - **Spark SQL**: Permite trabajar con datos estructurados y semi-estructurados, con lo que permite utilizar sintaxis SQL. Para este caso se tendrá que trabajar con DataFrames [3]
 - **Spark Streaming**: Capa encargada del procesamiento de flujo de datos, se puede manipular el flujo de manera similar a la manipulación de RDDs [16].

- **MLlib:** Biblioteca de ML (machine learning) que proporciona Spark. Cabe remarcar que actualmente la librería MLlib (la cual trabajaba con RDDs) a evolucionado a ML esta última está trabajando con DataFrames aunque hay que destacar que para la utilización de algunas métricas y diferentes transformaciones es necesario trabajar con MLlib.

Contiene una amplia opción de algoritmos tanto de aprendizaje supervisado como no supervisado. Mas adelante en los capítulos posteriores veremos en profundidad algunos de ellos.

- **GraphX:** Capa de procesamiento, manipulación sobre grafos y redes.

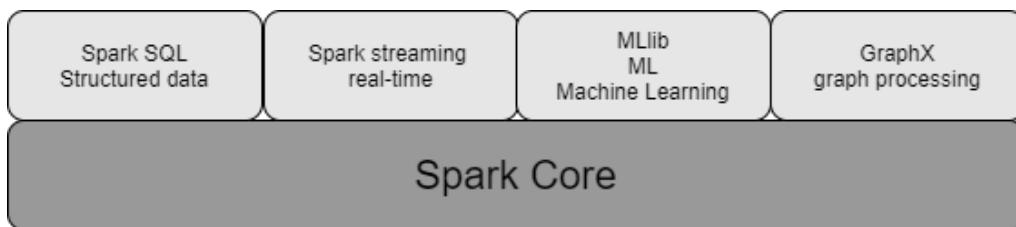


Figura 4.1: Estructura componentes Spark

Arquitectura Spark

En este apartado se va hacer un breve resumen sobre su arquitectura y los componentes más relevantes de ella. Como se ha comentado previamente Spark procesa datos en paralelo, para ello Spark utiliza una arquitectura basada en master/slave, donde tiene un master node y muchos slave workers nodes. Entrando en detalle en la arquitectura de Spark tenemos (ver Figura 4.2):

- **Driver program:** Sería el master node. Proporciona el código que Spark va a ejecutar en los workers que estan en un cluster. Crearía el SparkContext y negociaría como planificar los jobs (trabajos) con el Cluster manager la ejecucion de los jobs que posteriormente serán ejecutados en los workers (nodos).
- **Cluster manager:** Es el responsable de los recursos necesarios en los workers.
- **Executor:** Existiría solo en la ejecución de la aplicación y sería una JVM de Spark crea en cada worker.

- **Task:** Unidades de trabajo enviadas al Executor. Los Tasks contienen el resultado que son enviados de vuelta hacia el Driver o se puede redistribuir(shuffle) a otros workers dependiendo de la operación.

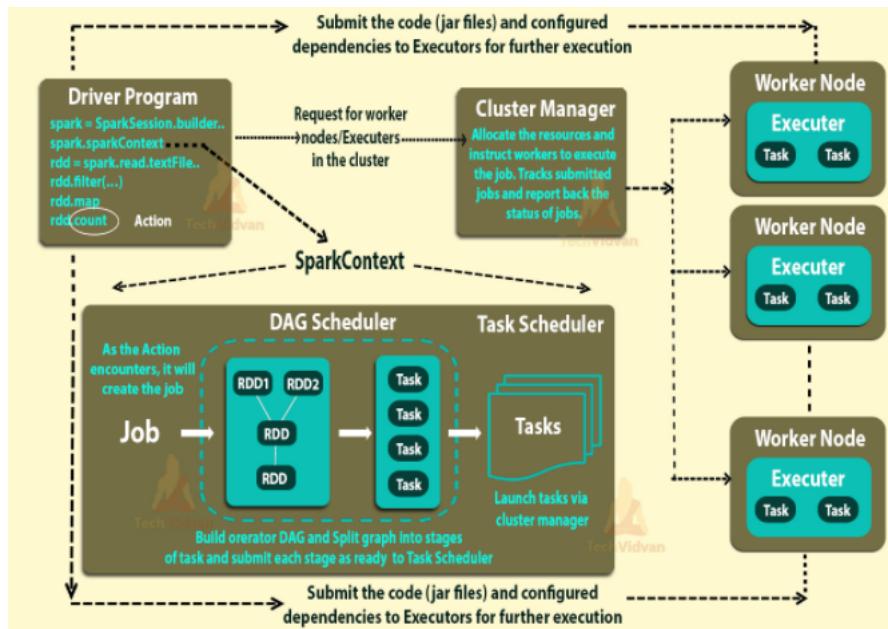


Figura 4.2: Arquitectura Spark (alto nivel)[1]

4.2. Spark ML/MLlib

Spark contiene librerías de ML las cuales trabajan con aprendizaje supervisado y no supervisado, principalmente estas librerías contienen los principales algoritmos para ambos casos (Supervisado y No supervisado). Por otro lado, gracias a la escalabilidad de Spark los métodos utilizados son totalmente escalables y fáciles de utilizar y configurar, también contienen herramientas para:

- **Featurización** en la librerías de ML de Spark. Trabajan con valores numéricos sin nulos y con dos columnas features (vector de los atributos del conjunto de datos) y labels (las clases del conjunto de datos).
- Creación de **pipelines** esta opción solo es válida para ML Spark.
- Opciones de **persistencia**.

Metric	Definition
Precision (Positive Predictive Value)	$PPV = \frac{TP}{TP+FP}$
Recall (True Positive Rate)	$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$
F-measure	$F(\beta) = (1 + \beta^2) \cdot \left(\frac{PPV \cdot TPR}{\beta^2 \cdot PPV + TPR} \right)$
Receiver Operating Characteristic (ROC)	$FPR(T) = \int_T^\infty P_0(T) dT$ $TPR(T) = \int_T^\infty P_1(T) dT$
Area Under ROC Curve	$AUROC = \int_0^1 \frac{TP}{P} d\left(\frac{FP}{N}\right)$
Area Under Precision-Recall Curve	$AUPRC = \int_0^1 \frac{TP}{TP+FP} d\left(\frac{TP}{P}\right)$

Figura 4.3: Métricas en Spark MLlib [12]

- Cálculos de **métricas** (*accuracy, PR, F1Score, AUC...*).

Con todo esto en Spark, para realizar aprendizaje automático en Spark, se pueden usar dos librerías:

- **Spark MLlib:** Basada en RDDs [16] actualmente está en un estado de mantenimiento donde no se desarrollan nuevas funcionalidades.
- **Spark ML:** Es la nueva librería (más moderna) de Machine Learning de Spark basada en DataFrames [3]. Por otro lado, Spark ML permite trabajar con el concepto de pipeline con lo que los conceptos principales serían (Figura: 4.4):
 - **DataFrame:** Como se ha descrito previamente esta API trabaja con este tipo de datos (DataFrames)
 - **Transformer:** Nos permite convertir un Dataframe en otro Dataframe, como modo de ejemplo es cuando hacemos una transformación de features a predicciones en un modelo ML
 - **Estimator:** Es un estimador el cual a partir de un `fit()` genera un transformer, como modo de ejemplo el estimador lo utilizaríamos para entrenar un conjunto de datos y generar un modelo el cual este último sería un transformer.

- **Pipeline:** Es el workflow donde juntamos varios estimadores y transformadores en serie.

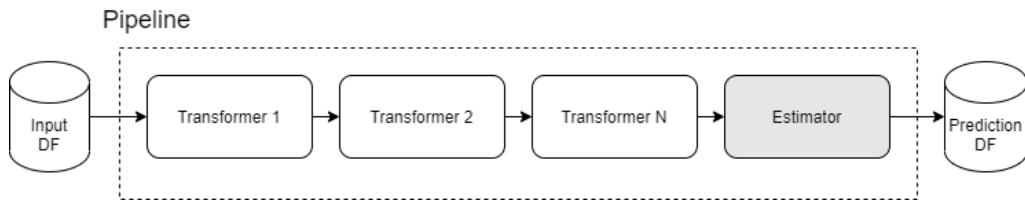


Figura 4.4: Workflow y pipeline

Este trabajo se va a focalizar principalmente en la librería de Spark ML trabajando con DataFrames. Por otro lado, a continuación se van a describir los diferentes algoritmos para el aprendizaje supervisado que Spark contiene en su librería ML principalmente nos vamos a focalizar en los clasificadores que son el core de los algoritmos semisupervisados y concretamente en los clasificadores probabilísticos (ver la Tabla: 5.4) que son los que se utilizarán en los experimentos descritos en los apartados siguientes.

4.3. DataBricks

Como herramienta de implementación y como infraestructura para ejecutar Spark basada en la nube se va a utilizar DataBricks.

DataBricks es una plataforma analítica escalable basada en la nube que permite trabajar con Spark tanto en Scala como en Python en un entorno de ejecución basado en Notebooks. Alguno de los principales beneficios serían:

- **Software as a service (SaaS)** con lo que no es necesario instalación
- **Fácil integración** con diferentes fuentes de datos, base de datos **SQL** y **NoSQL**, diferentes formatos, como **csv**, **json**, **xml...** y también fácil conexión soluciones de visualización entre otras.
- Fácil ejecución de **librerías Spark** ya embebidas en la plataforma con lo que hace muy simple la implementación y la experimentación
- **Multi-lenguaje:**
 - Scala

- Python
 - Java
 - R
- **Escalable** basado en la arquitectura Spark [4.1](#) trabajando con la versión
 - Librerías y Frameworks:
 - Ecosistema Spark
 - TensorFlow
 - Keras
 - pytorch
 - Scikit learn
 - Konda
 - XGBoost
 - **IDEs.** A parte del propio espacio de desarrollo que ofrece DataBricks. DataBricks ofrece la opción de poder trabajar con otros IDEs como:
 - RStudio
 - Eclipse Foundation
 - Visual Studio
 - Jupyter
 - Integración en la nube con Azure

A continuación se va hacer referencia a la configuración de DataBricks utilizada para este proyecto. Como proveedor de la nube se va a trabajar con Azure, de esta manera tendremos DataBricks como un servicio de Software (SaaS) sin necesidad de procesos de instalación haciendo muy fácil la puesta en marcha.

- Solución en la nube: Azure
- Lenguaje: Scala 2.12
- Spark 3.0.1
- Arquitectura Spark Seleccionado en Azure: Standard, caracterisitcas

- Worker - Standard DS3 v2 - 14GB Memory, 4 Cores, 0.75 DBU
- Driver - Standard DS3 v2 - 14GB Memory, 4 Cores, 0.75 DBU
- Numero de workers:
 - Min workers: 2
 - Max workers: 8

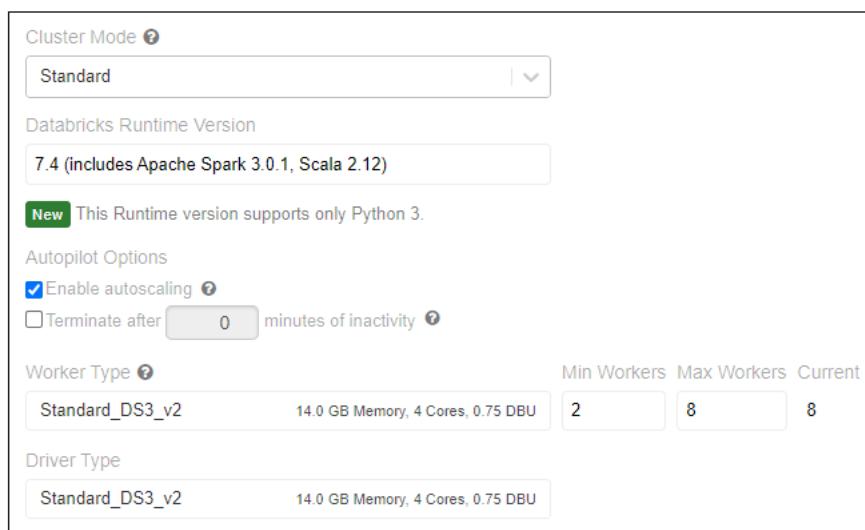


Figura 4.5: Configuración del cluster de ejecución en DataBricks

4.4. KEEL

Dado que en este trabajo se van a diseñar algunos algoritmos semi-supervisados utilizando Spark ML (apartado: 5) para su comprobación y verificación se van a comparar con los algoritmos de KEEL. KEEL ([7]) es una herramienta (open-source) basada en Java la cual contiene diferentes algoritmos tanto supervisados, no supervisados y semisupervisados (Figura: 4.6), esta herramienta esta pensada para ser utilizada tanto para la educación como para la investigación. En la figura 4.6 se ha remarcado en rojo los métodos que vamos comparar con nuestros resultados con Spark:

- semisupervisado - Single Classifier
- semisupervisado - Multi Classifier

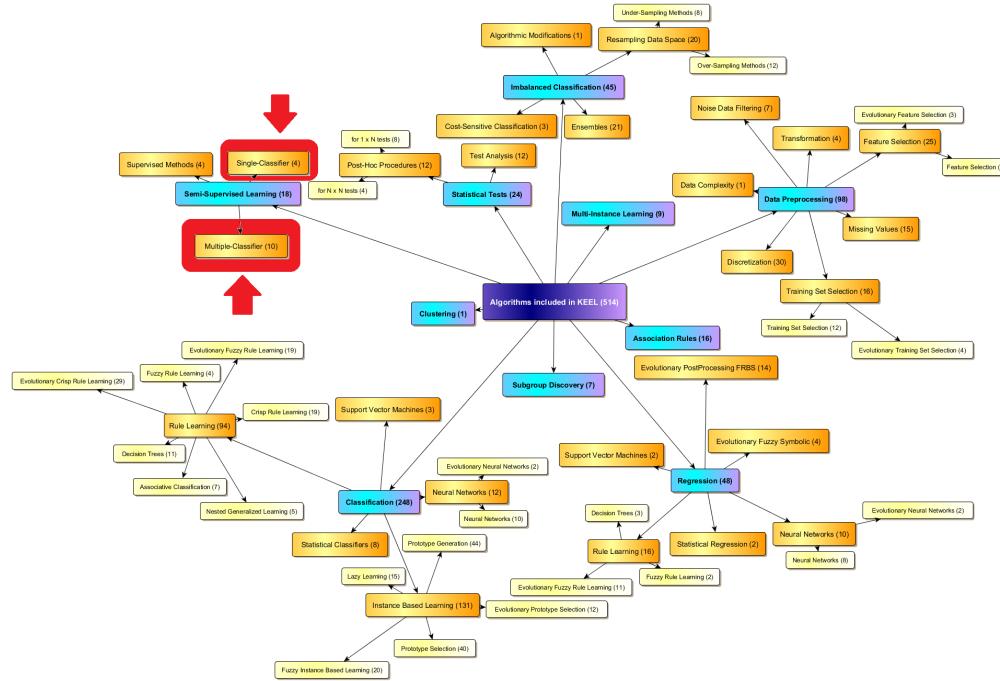


Figura 4.6: Métodos implementados en Keel [7]

4.5. Power BI

Para la representación de los resultados se va a utilizar Power BI.

Power BI es una herramienta de inteligencia de negocio de Microsoft la cual permite una infinidad de conexiones tanto a nivel local como en la nube con diferentes fuentes de datos, como modo de ejemplo se puede conectar con *SQL Server*, *Cosmos DB*, *SAP HANA*... Así como diferentes formatos como *csv*, *JSON*.... Power BI contempla dos tipos de soluciones:

- Power BI Desktop: Trabaja en local con una previa instalación permite diseñar dashboards pero no publicarlos ni compartirlos online (distribución gratuita)
- Power BI Publisher: Permite compartir online los dashboards a partir de una aplicación web en este caso esta solución se puede adquirir bajo pago.

4.6. Python SeaBorn y Matplotlib

Para alguna visualizaciones especialmente para el capítulo 5 se ha utilizado Python, concretamente la versión 3 con las librerías:

- **Matplotlib**: Principal librería de Python para representación gráfica, la primera que se creó y utilizada en las otras librerías de visualización de Python.
- **SeaBorn**: Utiliza Matplotlib de base, fácil de utilizar y implementar sin la necesidad de un gran desarrollo, también hay una mejora estética respecto a Matplotlib.

Capítulo 5

Aspectos relevantes del desarrollo del proyecto

En este apartado se va hacer referencia a la propuesta de estudio. Para ellos nos hemos basado en el artículo [32] donde se hace una comparativa muy extensa entre diferentes algoritmos semisupervisados tanto trabajando con métodos transductivos como inductivos y con conjunto de datos con clase múltiple y binaria.

Este trabajo se va a centrar en trabajar con métodos inductivos [33] y con conjuntos de datos de clase binaria.

Por otro lado, después de analizar los resultados para los métodos inductivos semisupervisados del artículo [32] nos hemos centrado en sus mejores resultados los cuales han sido obtenidos por los siguientes métodos sin tener en cuenta los clasificadores base utilizados (Tabla: 5.1)

Semi-Supervised	View	Learning	Classifiers	Processing	Algorithm
Self Labeled	Single View	Single Learning	Single Classifier	Batch	Self Training
			Muti Classifier	Batch	TriTraining Co-Trainning CoBagging LMT
		Multi Learning	Multi Classifier	Batch	Democratic-Co

Tabla 5.1: Algoritmos semisupervisados con mejores resultados según el artículo [32].

Finalmente para su implementación, diseño y posterior estudio de este trabajo se han seleccionado el Self-Training y el Co-Training con los clasifi-

cadores base probabilísticos más comunes en Spark, tal y como resume la Tabla: 5.2.

Semi-Supervised	View	Learning	Classifiers	Processing	Algorithm	Base Classifier (Spark)
Self Labeled	Single View	Single Learning	Single Classifier	Batch	Self Training	Logistic Regression Decision Tree Naive Bayes Random Forest
			Muti Classifier	Batch	Co-Training	Logistic Regression Decision Tree Naive Bayes Random Forest

Tabla 5.2: Algoritmos semisupervisados y clasificadores base en Spark seleccionados.

5.1. Datos utilizados

En esta sección se van a presentar los datos utilizados para el diseño y los experimentos, para ello se han seleccionado diferentes conjuntos de datos. Por un lado se han utilizado 8 conjuntos de datos menores a $10K$ instancias y por otro lado superiores a $1M$ de instancias este último focalizado para un perfil de datos Big data donde Spark tiene una importancia más relevante que otras tecnologías que no trabajan en memoria (sino en disco) y no son distribuidas. Por otro lado, en este trabajo nos vamos a focalizar en datos con clase binaria dejando para posibles estudios futuros datos multi-clase.

5.2. Algoritmos supervisados - Clasificadores base

Como se ha explicado previamente en el capítulo 3.3, los algoritmos de clasificación supervisada trabajan con uno o varios clasificadores base

Conj. de Datos	Nº de insts.	Nº de atr.	Real/Int/Nominal	Clases Categóricas	Distr. de Clase label 0 (%)	Distr. de Clase label 1 (%)
sonar	208	60	60-0-0	2	53.4	46.6
spectfheart	267	44	0-44-0	2	79.4	20.6
heart	270	13	1-12-0	2	55.5	44.5
wisconsin	699	2	0-9-0	2	65.0	35.0
Titanic	2201	3	3-0-0	2	67.7	32.3
Banana	5300	2	2-0-0	2	55.1	44.9
coil2000	9822	85	0-85-0	2	95.0	5.0
magic	19020	10	10-0-0	2	64.8	35.2
Adult	48842	14	0-0-14	2	75.0	25.0
Poker	1025010	11	0-5-6	2 (*)	50.12	49.88
TaxiNY	1458644	11	6-3-1	2 (*)	66.4	33.6

Tabla 5.3: Datos utilizados en el diseño y en los experimentos (*después del pre-procesado)

en función del algoritmo utilizado. Para este trabajo se van a utilizar los algoritmos más comunes en Spark ML definidos en la tabla 5.4. Estos clasificadores serán introducidos como parámetro para los algoritmos SSC.

Abreviatura	Clasificador	referencia Spark
DT	Decision Tree Classifier	[4]
LR	Logistic regression	[11]
NB	Naive Bayes	[13]
RF	Random Forest	[15]

Tabla 5.4: Clasificadores base métodos

5.3. Algoritmos semi-supervisados implementados

En este trabajo como se ha explicado previamente se van a diseñar los métodos Self-Training y Co-Training en Spark ML (Tabla: 5.2). En esta sección se va a explicar cada uno de los algoritmos:

Self-Training

Basado en un clasificador base y Single view (no habrán particiones de datos en los entrenamientos), para ellos se seguirá el algoritmo definido en [21] y [36].

Algorithm 1 Algoritmo Self-Training con umbral/threshold

Require: $L, U, m, Threshold$

```

1: repeat
2:   Entrenamos  $L$  con el clasificador base y creamos el modelo  $m$ 
3:   for  $x \in U$  do
4:     if  $\max m(x) > Threshold$  then
5:        $L \leftarrow L \cup \{(x, pred(x))\}$ 
6:        $U \leftarrow U - \{x\}$ 
7:     end if
8:   end for
9: until no hay más predicciones fiables or se ha superado el número
máximo de iteraciones

```

En el algoritmo 2 se trabaja con criterio kBest donde en vez de añadir los datos no etiquetados en función de un umbral/threshold (definido en el algoritmo 1) se etiqueta una cantidad definida por kBest de las mejores probabilidades de predicción de las clases no etiquetado D.4.

Algorithm 2 Algoritmo Self-Training con kBest

Require: $L, U, m, kbest, L_{new}$

```

1: repeat
2:   Entrenamos  $L$  con el clasificador base y creamos el modelo  $m$ 
3:   while  $count(L) <= kBest$  do
4:     Donde  $x \in U$ 
5:     if  $\max m(x) > Threshold$  then
6:        $L \leftarrow L \cup \{(x, pred(x))\}$ 
7:        $U \leftarrow U - \{x\}$ 
8:     end if
9:   end while
10:  until no hay mas predicciones fiables or se ha superado el número
      máximo de iteraciones

```

Por último también podemos ver la descripción del algoritmo Self-Training (1) de una forma gráfica en la Figura 5.1

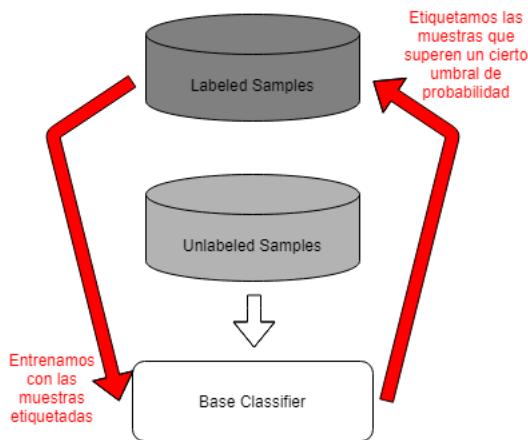


Figura 5.1: Ilustración funcionamiento algoritmo Self-Training.

Co-Training

Basado en multi view donde se va hacer una partición en sus datos de entrenamiento donde tendríamos los datos etiquetados y no etiquetados por

otro lado utiliza dos clasificadores (Multiple classifier) Es decir, se tendrá un clasificador para un grupo de datos no etiquetados y otro para el grupo restante. Posteriormente este clasificador etiquetará cada uno de los grupos no etiquetados en cada una de las iteraciones (Observar Algoritmo: 3, Figura: 5.2 y la Tabla: 3.1)

Algorithm 3 Algoritmo Co-Training con umbral/thresholds

Require: $L, U, m_1, m_2, Threshold, L_1, U_1, L_2, U_2$ donde se genera una partición en $L \rightarrow L_1, L_2$ y en $U \rightarrow U_1, U_2$

```

1: repeat
2:   Entrenamos  $L_1$  y  $L_2$  con sus clasificadores base y creamos el modelo
    $m_1$  y  $m_1$ 
3:   for  $x \in U_1$  do
4:     if  $\max m_1(x) > Threshold$  then
5:        $L_2 \leftarrow L_2 \cup \{(x, pred_1(x))\}$ 
6:        $U_1 \leftarrow U_1 - \{(x)\}$ 
7:     end if
8:   end for
9:   for  $x \in U_2$  do
10:    if  $\max m_2(x) > Threshold$  then
11:       $L_1 \leftarrow L_1 \cup \{(x, pred_2(x))\}$ 
12:       $U_2 \leftarrow U_2 - \{(x)\}$ 
13:    end if
14:  end for
15: until no hay mas predicciones fiables or se ha superado el número
   máximo de iteraciones

```

A continuación se especifica el algoritmo: 4 Co-Training con kBest, Observar la Tabla: D.5 y con threshold, algoritmo: 3

Como se ha hecho en la explicación del Self-Training se ha añadido una imagen donde se puede apreciar el comportamiento del algoritmo Co-Training en la Figura: 5.2

5.4. Verificación y test de los algoritmos SSC implementados

En la siguiente sección se va a realizar la verificación de los algoritmos diseñados en este trabajo, Self-Training y Co-Training con Spark, para ello utilizaremos los resultados obtenidos en Keel, concretamente trabajando

Algorithm 4 Algoritmo Co-Training con kBest

Require: $L, U, m_1, m_2, Threshold, L_1, U_1, L_2, U_2, L_{new1}, L_{new2}$ donde se genera una partición en $L \rightarrow L_1, L_2$ y en $U \rightarrow U_1, U_2$

- 1: **repeat**
- 2: Entrenamos L_1 y L_2 con sus clasificadores base y creamos el modelo m_1 y m_1
- 3: **while** $count(L_2) < kBest/2$ **do**
- 4: Donde $x \in U_1$
- 5: **if** $\max m_1(x) > Threshold$ **then**
- 6: $L_2 \leftarrow L_2 \cup \{(x, pred_1(x))\}$
- 7: $U_1 \leftarrow U_1 - \{(x)\}$
- 8: **end if**
- 9: **end while**
- 10: **while** $count(L_1) < kBest/2$ **do**
- 11: Donde $x \in U_2$
- 12: **if** $\max m_2(x) > Threshold$ **then**
- 13: $L_1 \leftarrow L_1 \cup \{(x, pred_2(x))\}$
- 14: $U_2 \leftarrow U_2 - \{(x)\}$
- 15: **end if**
- 16: **end while**
- 17: **until** no hay mas predicciones fiables *or* se ha superado el número máximo de iteraciones

con alguno de los conjuntos de datos utilizados en los artículos [32] y [21] y con el algoritmo de árbol de decisión (Decision Tree - DT) como clasificador base ya que es el algoritmo supervisado que tenemos en común entre los artículos antes mencionados utilizando Keel y Spark. Con todo esto vamos a realizar la verificación de nuestros algoritmos de la siguiente forma:

1. Analizaremos solo el clasificador base en Keel y en Spark (la solución por defecto) sin reducción de datos y compararemos los resultados para entender su comportamiento y los futuros comportamientos de los algoritmos SSC.
2. Analizaremos el comportamiento del Self-Trainig y el Co-Training, a continuación definiremos los parámetros para los métodos utilizados:
 - a) **Clasificador base:** DT (como se ha explicado previamente)
 - b) **Criterio** (definición: D.1 y D.5): Threshold
 - c) **Threshold:** 0.9

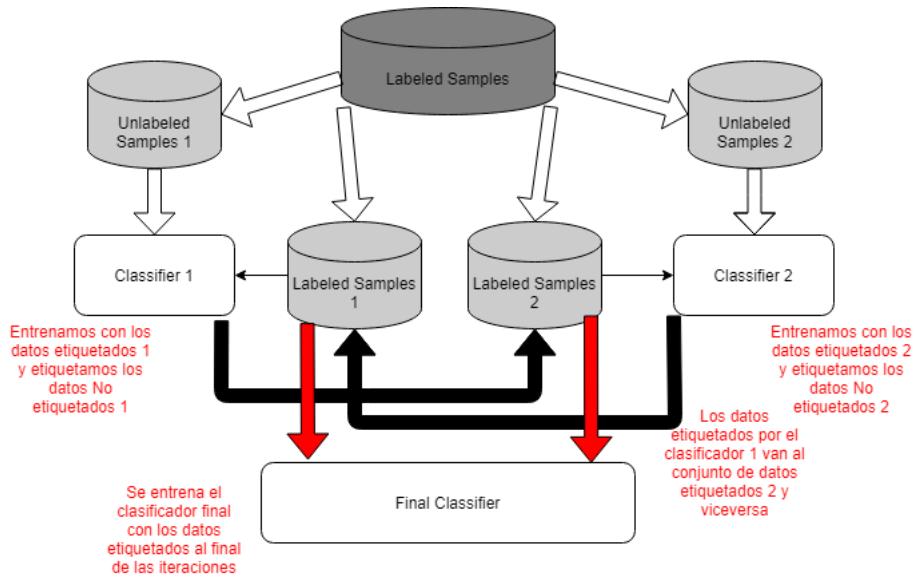


Figura 5.2: Ilustración funcionamiento algoritmo Co-Training.

- d) **Validación cruzada** (definición D.6): 10
- e) **Porcentaje de etiquetado**: 0.1, 0.2, 0.3 y 0.4
- f) **Métodos SSC**: Self-Training y Co-Training

Clasificador Base (DT) Keel vs Spark

Para este apartado como se ha explicado previamente se han seleccionado 8 conjuntos de datos (*wisconsin*, *titanic*, *spectheart*, *sonar*, *magic*, *heart*, *coil2000*, *banana*), tabla 5.3, los cuales se pueden encontrar en [7] por otro lado, en la Figura 5.3 se muestra el comportamiento del clasificador base cuando desetiquetamos las instancias para el conjunto de datos sonar para ello se han utilizado las métricas, *accuracy*, *PR*, *UAC*, *F1Score*.[12]

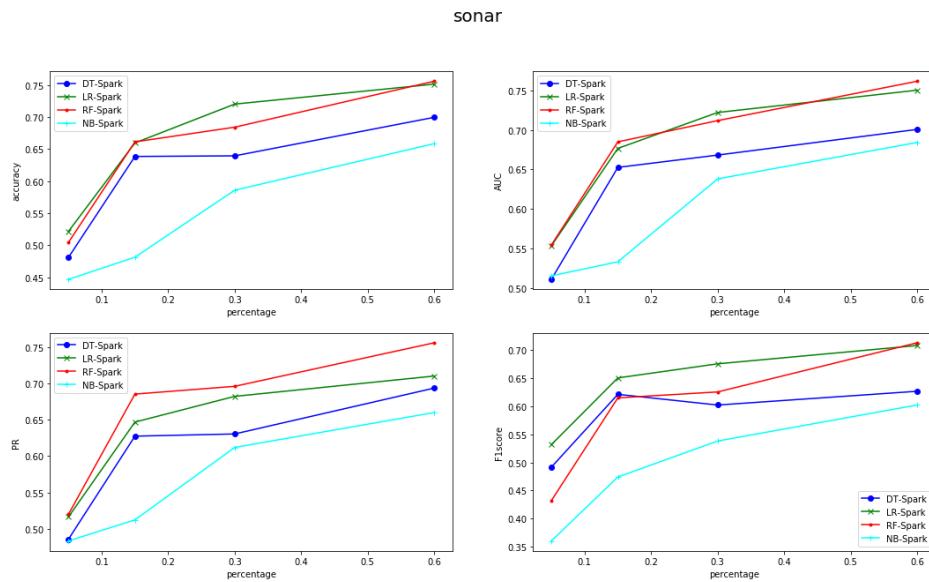


Figura 5.3: Resultados clasificadores base (supervisados) desetiquetando instancias

Se puede ver que cuantas más instancias etiquetadas tiene mejor es el resultado como es de esperar, por otro lado el resultado entre todas las métricas es similar ya que las clases están bastante bien balanceadas (excepto para el conjunto de datos coil200), como pasa en todo los conjuntos de datos (Tabla 5.3).

Finalmente hemos utilizado *accuracy* dado que los artículos [32] y [21] (trabajando con Keel) que estamos utilizando para comparar nuestros resultados utilizan dicha métrica.

Por último, se puede ver (Figura: 5.4) que el clasificador base (DT) por defecto tanto en Keel como en Spark tienen un resultado similar. Hay que tener en cuenta que los resultados de Keel son diferentes ya que Keel utiliza el algoritmo de árbol de decisión C4.5 y Spark trabaja con una combinación

de ID3 con CART, aparte de eso, Keel hace post-poda después de generar el modelo sin límite de profundidad en cambio Spark tienen una profundidad editable (5 por defecto, la cual puede ser modificada) todo esto hace que los resultados para Keel sean ligeramente superiores con sus parámetros por defecto para 5 de los 8 conjunto de datos utilizados, como se ve en la Figura 5.4 (el caso más desfavorable sería para el conjunto de datos sonar con un 0,3 la diferencia de *accuracy*), posiblemente si utilizáramos otros parámetros en Spark podríamos mejorar los resultados del mismo, pero el objetivo de este apartado es verificar si la implementación del los algoritmos Self-Training y Co-Training en Spark tienen el comportamiento esperado y ese comportamiento puede ser derivado del clasificador base, por eso necesitamos entender el comportamiento del mismo antes de adentrarnos en los métodos SSC.

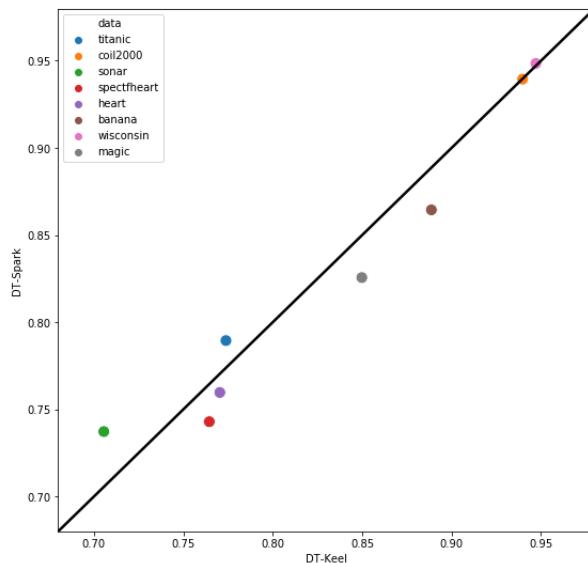


Figura 5.4: Resultados Keel vs Spark Decision Tree (DT) supervisado

Self-Training con DT Keel vs Spark

En este apartado se va hacer la comparativa entre los resultados con los conjuntos de datos mencionados anteriormente 5.4 y entre el método Self-Training con Spark y el mismo con Keel con el objetivo de entender si la implementación de este método en Spark es correcta, para ello es importante entender los resultados o las diferencias encontradas en el clasificador base utilizado (DT, 5.4) para ver si las diferencias para el Self-Training Spark y Keel también siguen una tendencia similar al clasificador base.

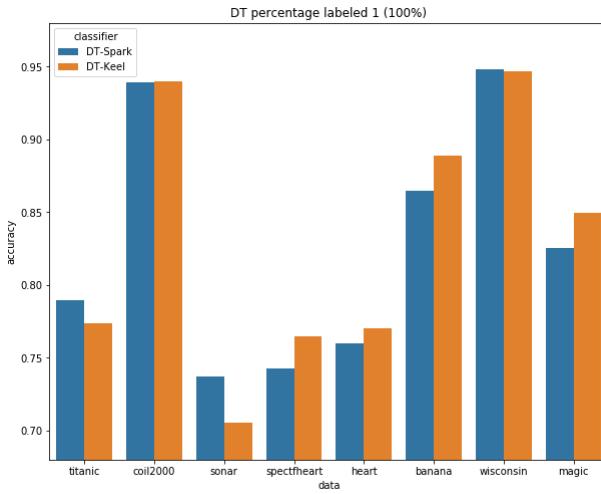


Figura 5.5: Resultados Keel vs Spark, Supervisado Decision Tree/ árbol de decisión (DT) como clasificador base, con todos los datos etiquetados

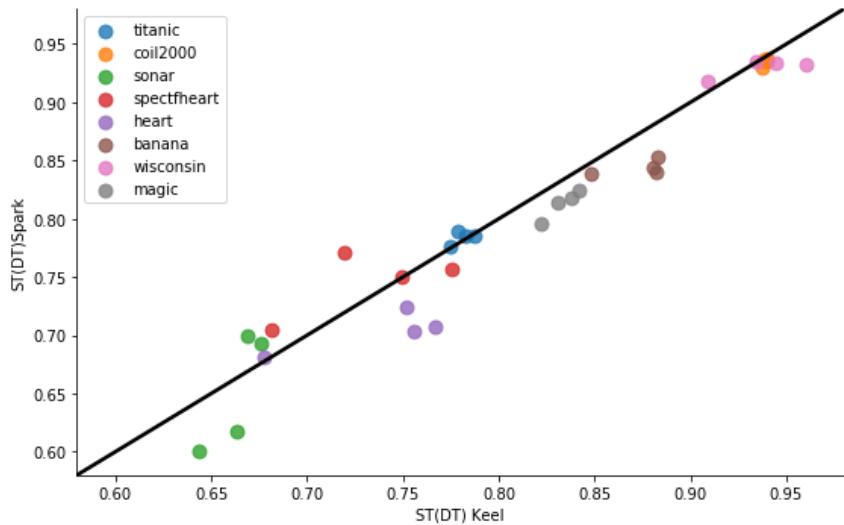


Figura 5.6: Resultados Keel vs Spark, Self-Training (ST) con DT como clasificador base ST (DT)

Para ello vamos a estudiar el Self-Training con las características/parámetros mencionados en 5.4 punto 2. Como se puede ver en la Figura: 5.6 se representa los datos de la accuracy entre los métodos de Self-Training con

Keel y Spark podemos observar que la tendencia está siguiendo la diagonal con lo cual los resultado serían bastante similares con la diferencia propia proveniente de los clasificadores base. En las gráficas de barras de Figura (5.7) se puede apreciar en más detalle donde la diferencia máxima es de 0.3 aproximadamente para el conjunto de datos sonar.

Con todo lo antes mencionado podemos decir que la implementación del Self-Training en Spark es correcta incluso en algunos casos como el del conjunto de datos heart se ha conseguido tener una mejora en resultados (concretamente trabajando con una porcentaje de etiquetado del 10%) partiendo de un resultado peor para el clasificador base. Si se hace una comparativa entre el método supervisado en Keel, Spark (Figura: 5.4 o en gráfico de barras, Figura: 5.5) y posteriormente Self-Training de Keel y Spark se puede observar que hay muchos mas puntos por encima de la diagonal, con lo cual se puede determinar que la implementación en Spark mejora en general el comportamiento partiendo con un peor clasificador base para el método Self-Training.

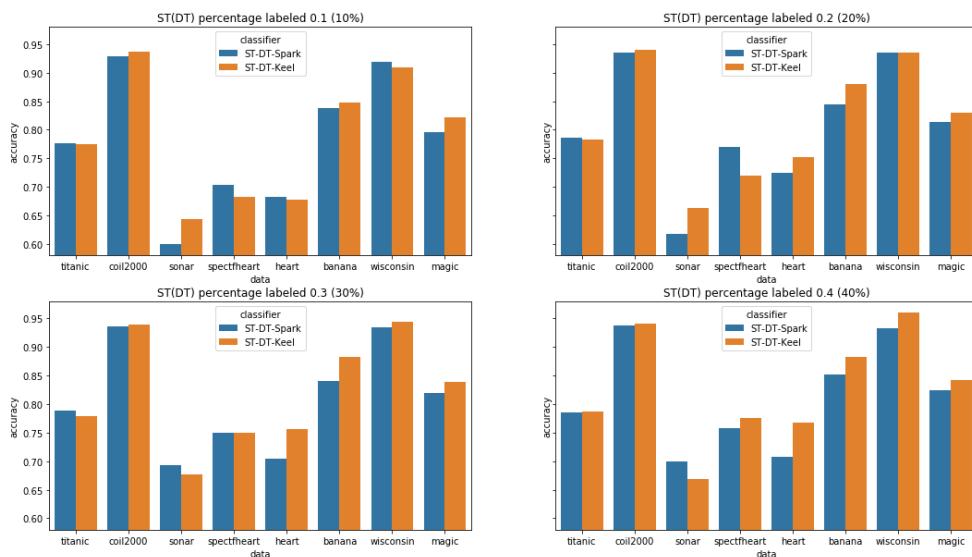


Figura 5.7: Resultados Keel vs Spark, Self-Training con DT como clasificador base, cada gráfico representa un porcentaje de etiquetado

Co-Training con DT Keel vs Spark

Siguiendo la misma idea que el apartado ?? estudiaremos el comportamiento con Co-Training en Keel y Spark según el artículo [32] y [21] con la configuración mencionada en 5.4 punto 2.

Como se puede apreciar en la Figura 5.8 también seguiría la diagonal con lo cual se asimilaría bastante al Co-Training de Keel, cabe remarcar que habría más diferencia entre ellos (Co-Training Spark vs Co-Training Keel) sobretodo en el conjunto de datos heart y sonar, también se puede apreciar que si comparáramos con el caso Supervisado (Figura 5.4) el caso Co-Training tiene un resultado bastante bueno ya que partiendo con un clasificador base peor (Figura 5.4), se consigue tener también casos por encima de la diagonal con lo cual tendríamos un rendimiento en segín que casos mejor con Spark incluso partiendo con clasificador base peor. También se puede observar el mismo resultado pero en un gráfico de barras en la Figura: 5.9.

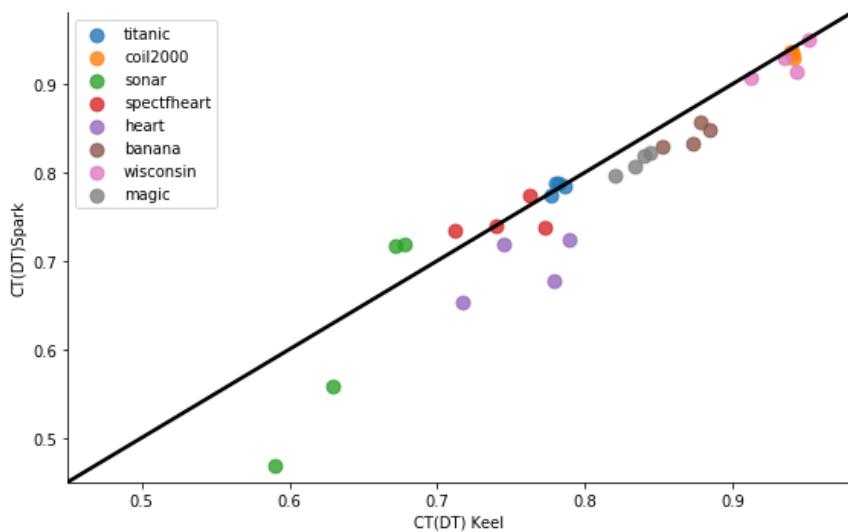


Figura 5.8: Resultados Keel vs Spark, Self-Training (CT) con CT como clasificador base ST(DT)

Diseño y Experimentación

Una vez se a verificado que los métodos diseñados en Spark están trabajando correctamente, en el Apéndice C se presenta el diseño realizado donde se explica el tipo de pipeline utilizado, sus estados así como la arquitectura

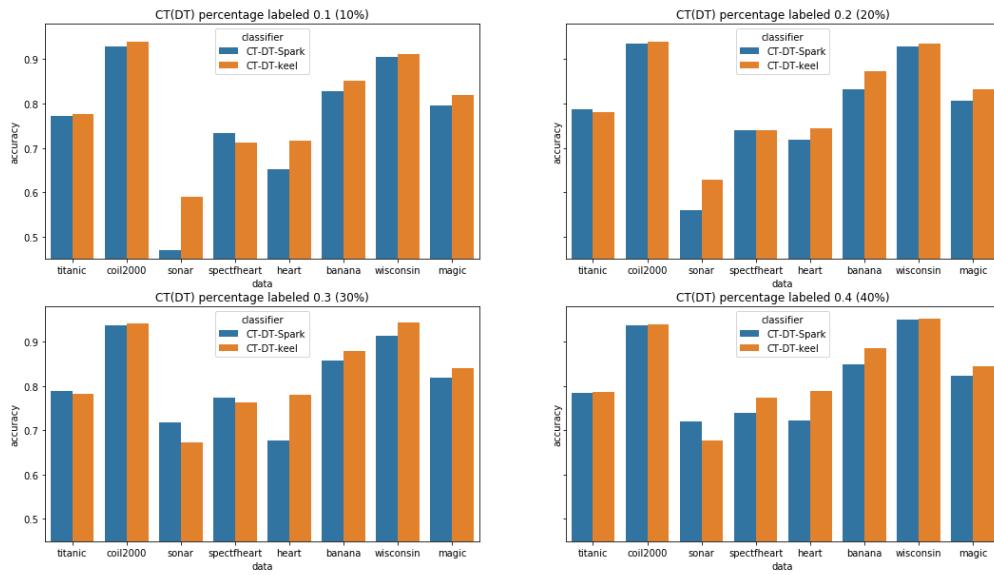


Figura 5.9: Resultados Keel vs Spark, Co-Training con DT como clasificador base, cada gráfico representa un porcentaje de etiquetado

utilizada. Por otro lado en el Apéndice B se presenta los resultados entre los diferentes métodos (Self-Training y Co-Training), la comparativa entre ellos y con sus clasificadores base (NB, LR, RF, DT).

Capítulo 6

Trabajos relacionados

Como se ha descrito en los capítulos previos, especialmente en el capítulo 5 se ha utilizado como base de este trabajo el artículo [32] el cual hace una comparativa de los diferentes métodos (observar la Figura 3.4) de clasificadores semisupervisados (SSC) tanto para los algoritmos transductivos como inductivos. Focalizándose en una comparativa entre ellos trabajando con KEEL (Figura 4.6, [7]), la herramienta esta escrita en Java, con lo cual su arquitectura es una arquitectura vertical (no distribuida) por tanto va a tener una performance en tiempo y en coste mucho peor que si trabajamos en memoria con una arquitectura horizontal (distribuida). Por otro lado, dado que actualmente los métodos de SSC están en auge (Figura: 3.2) se han encontrado muchos trabajos donde se explican los métodos más comunes para SSC (observar la Tabla 3.1). También se han encontrado otros trabajos interesantes trabajando con SSC en Spark, entre ellos se ha seleccionado [26], [31].

El artículo [26] hace un estudio sobre SSL de sentimientos trabajando con la API de twitter para adquirir los datos y concretamente con los algoritmos de Self-Trainning y Co-Training en batch y streaming en Spark, dado que en este trabajo se trabaja en Batch nos hemos focalizado en este tipo de procesamiento. Como puntos interesantes, este artículo hace una comparativa cuando los datos no están bien balanceados y como se muestra en la Figura 6.1 extraída del mismo artículo [26] se ve que con Co-Training se mejora el overfitting y se consigue un balance de predicción de clase mucho mejor en el caso del experimento, aumentando por dos la predicción de sentimientos negativos.

Batch annotation with Co-Training: Annotated results per class for different confidence values δ .

δ	positive predictions	negative predictions	unlabeled
65%	175,704,567 (76.64%)	53,547,561 (23.35%)	0.66%
70%	178,361,861 (78.20%)	49,544,245 (21.73%)	1.25%
75%	181,019,145 (79.75%)	45,119,636 (20.25%)	2.15%
80%	182,189,488 (81.22%)	41,229,89 (18.47%)	3.17%
85%	182,758,594 (83.04%)	37,300,75 (16.95%)	4.65%
90%	182,707,849 (85.06%)	32,069,220 (14.93%)	6.93%
95%	179,527,239 (87.43%)	25,810,953 (12.56%)	11.02%
100%	1,281,748 (99.69%)	5,116 (0.39%)	99.44%
Initial Model	2,211,091 (87.47%)	316,662 (12.52%)	

Batch annotations with Self-Learning: Annotated results per class for different confidence values δ .

δ	positive predictions	negative predictions	unlabeled
65%	201,860,127 (88.46%)	26,315,693 (11.53%)	1.13%
70%	200,212,418 (88.49%)	26,033,466 (11.50%)	1.97%
75%	198,296,101 (88.59%)	25,525,791 (11.40%)	3.02%
80%	196,017,401 (88.78%)	24,757,934 (11.21%)	4.34%
85%	190,130,256 (89.05%)	23,866,748 (10.95%)	6.08%
90%	189,271,802 (89.49%)	22,217,578 (10.50%)	8.36%
95%	183,012,328 (90.21%)	19,845,802 (9.78%)	12.10%
100%	650,450 (99.86%)	877 (0.13%)	99.71%
Initial Model	2,211,091 (87.47%)	316,662 (12.52%)	

Figura 6.1: Artículo [26] accuracy Self-Training y Co-Training

Por último, también se puede ver el comportamiento entre el Self-Training y el Co-Training respecto la accuracy con el porcentaje de datos etiquetados y no etiquetados. Para este caso concreto (Figura: 6.2) se puede apreciar que si los datos de entrenamiento son < 40 % el Self-Training tendría un mejor comportamiento. Para este artículo se trabaja con un único tipo de dato, de lo contrario para nuestro trabajo se ha buscado diferentes tipos de datos ya que de lo contrario es complicado sacar conclusiones concretas.

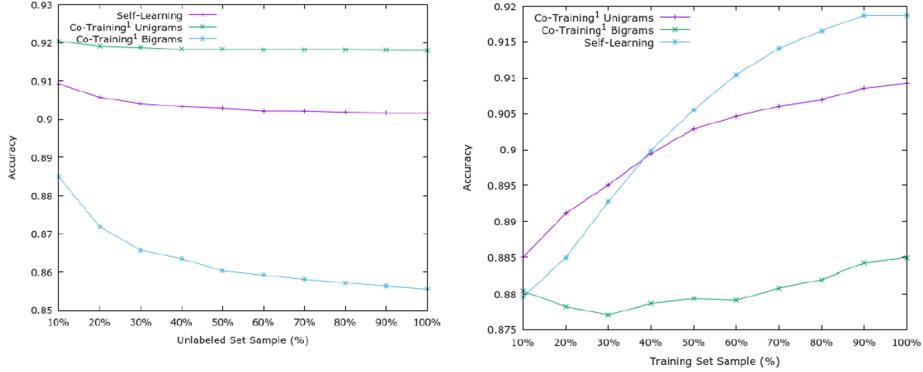


Figura 6.2: Artículo [26] Self-Training y Co-Training. Accuracy vs datos etiquetados y no etiquetados

En relación al otro artículo antes mencionado ([31]) se focaliza a entender mejor porque en muchas ocasiones SSC funciona peor que los métodos supervisados y cual es el mejor contexto para utilizar SSC. Principalmente menciona que hay tres tipos de motivos que influyen en gran medida en los métodos de SSC, entre ellos :

- **Distribución de clase:** Las distribución de clase tendría que ser similar entre los datos etiquetados y no etiquetados este último es complicado ya que no se dispone de la información de las etiquetas,

pero se podría extraer la información si los datos fueron extraídos el mismo día, en el mismo sitio (ejemplo la misma máquina para temas de IoT, sensores...) que los datos ya etiquetados, lo que nos viene a decir el artículo es que cuanto más “mismatch” entre los dos conjuntos de datos más error en el resultado final del modelo SSC (Figura: 6.3)

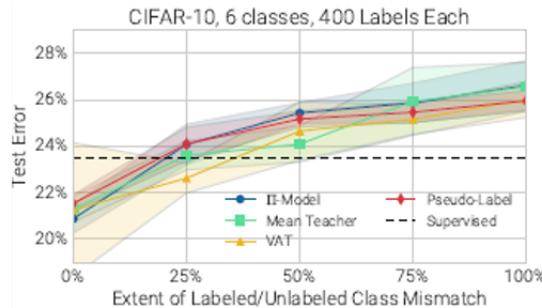


Figure 2: Test error for each SSL technique on CIFAR-10 (six animal classes) with varying overlap between classes in the labeled and unlabeled data. For example, in “25%”, one of the four classes in the unlabeled data is not present in the labeled data. “Supervised” refers to using no unlabeled data. Shaded regions indicate standard deviation over five trials.

Figura 6.3: Figura extraída del artículo [31] Comportamiento con diferentes distribuciones de clase

- **Conjunto de datos de validación:** Tener un equilibrio entre los datos de entrenamiento y validación (todos serían datos de entrenamiento los datos de validación no serían los de test) utilizados para la elección de los hyperparámetros utilizados para la configuración del modelo es muy importante ya que en ocasiones trabajar con más datos de validación encarece el proceso y no se obtienen mejorías en ocasiones puede empeorar el modelo, Figura 6.4 (normalmente es bueno utilizar un 10 % del modelo de entrenamiento aunque en algunos casos es complicado cuando hay un valor muy pequeño de datos etiquetados).
- **Relación con los tamaños de datos etiquetados y no etiquetados:** Es importante mantener una relación entre el tamaño de los datos etiquetados y no etiquetados, en muchas ocasiones por trabajar con más datos no etiquetados el resultado no es mejor, llega a estabilizarse, consume más recursos y para el caso que nos interesa self-labeled incluso empeora (Figura: 6.5).

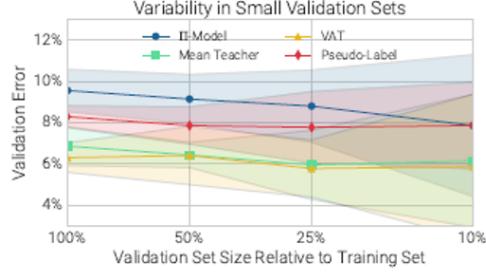


Figure 5: Average validation error over 10 randomly-sampled nonoverlapping validation sets of varying size. For each SSL approach, we re-evaluated an identical model on each randomly-sampled validation set. The mean and standard deviation of the validation error over the 10 sets are shown as lines and shaded regions respectively. Models were trained on SVHN with 1,000 labels. Validation set sizes are listed relative to the training size (e.g. 10% indicates a size-100 validation set). X-axis is shown on a logarithmic scale.

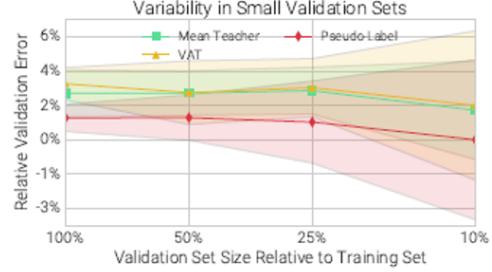


Figure 6: Average and standard deviation of relative error over 10 randomly-sampled nonoverlapping validation sets of varying size. The experimental set-up is identical to the one in fig. 5, with the following change: The mean and standard deviation are computed over the difference in validation error compared to II-model, rather than the absolute validation error. X-axis is shown on a logarithmic scale.

Figura 6.4: Figura extraída del artículo [31] Comportamiento con diferentes tamaños en los conjuntos de validación

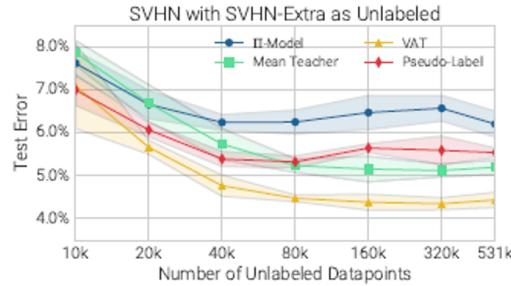


Figure 3: Test error for each SSL technique on SVHN with 1,000 labels and varying amounts of unlabeled images from SVHN-extra. Shaded regions indicate standard deviation over five trials. X-axis is shown on a logarithmic scale.

Figura 6.5: Figura extraída del artículo [31] Comportamiento con diferentes tamaños entre datos etiquetados y no etiquetados

Finalmente, también se han encontrado otros trabajos en Github los cuales han implementado métodos SSC ([8], [9]) que se han utilizado como referencia para este trabajo.

En ninguno de los trabajos/artículos encontrados se hace una comparativa con todos los clasificadores probabilísticos de Spark con los métodos Self-Training y Co-Training vs Supervisado en Spark como se ha realizado en este

trabajo ni se ha hecho un diseño totalmente parametrizado donde se puede utilizar de una manera muy fácil y simple sin necesidad de re-programar para la experimentación con Spark ML.

Capítulo 7

Conclusiones y Líneas de trabajo futuras

En este trabajo se han implementado, validado y posteriormente se han creado las librerías para los algoritmos Self-Training y Co-Training. Posteriormente se ha realizado una parte experimental con diferentes conjuntos de datos con la idea de hacer una comparativa con los algoritmos semisupervisados.

Por otro lado en este trabajo se ha podido demostrar que los métodos implementados en Spark ML funcionan correctamente y en ocasiones tienen un mejor resultado comparado con la herramienta KEEL (5) por otro lado, también se a podido demostrar que para diferentes conjuntos de datos los algoritmos semisupervisados tienen un resultado mejor que los algoritmos supervisados trabajando con un porcentaje de etiquetado concreto (B y B.7). Lo cual indicaría que tendría sentido el hecho de utilizar métodos semisupervisados.

Remarcar que los métodos se han integrado a la estructura de Spark como clases (D) y posteriormente se ha encapsulado en una biblioteca (*org.apache.spark.ml.semisupervised.*) a partir de la creación de un .jar como se puede ver en el apartado D.4, con lo cual es muy sencillo reutilizarlos para líneas de trabajo futuras o como guía para futuras implementaciones de nuevos métodos SSC en Spark ML.

Como posibles trabajos futuros, se han identificado los siguientes puntos:

- Repetir los experimentos con datos multi-clase.

- Repetir los experimentos con más conjunto de datos, para sacar conclusiones más concretas y poder identificar las mejores configuraciones.
- Repetir los experimentos con datos desbalanceados y ver si tienen resultados similares.
- Implementar otros algoritmos SSC (Tabla: 3.1) diferentes a los implementados en este trabajo (Self-Training y Co-Training).
- Buscar algún caso de uso real para poder implementar los algoritmos SSC y ver si en un caso real se consiguen las ventajas que se han conseguido en un entorno experimental.
- Repetir los experimentos con otro criterio diferente al Threshold en este caso utilizando kBest.
- Buscar patrones con un gran volumen de conjunto de datos en función de sus características donde se pueda realizar alguna guía de uso donde nos indique si utilizar semisupervisado o supervisado. (D.4 y D.5)
- Combinar Co-Training con diferentes clasificadores base.

Apéndices

Apéndice A

Plan de Proyecto

A.1. Introducción

En este apartado se va explicar con que herramientas de planificación/gestión y como se ha llevado acabo la implementación de este trabajo. Para ello y dado que se trabaja en remoto entre el autor y sus tutores se han utilizado herramientas online para compartir tanto la planificación, estado, desarrollo/trabajo realizado y la memoria. Se han utilizado las siguientes herramientas:

- **ZenHub:** [18] Herramienta de gestión de proyecto para aplicar metodología ágiles, la cual se ha utilizado en este trabajo tanto para definir las tareas (Issues) las cuales se pueden englobar o agrupar por grupo de tareas (Epics). ZenHub es una herramienta online gratuita, solo es necesario registrarse. Por otro lado, también se han realizado todas las consultas a partir de esta herramienta con los tutores de este trabajo, de esta manera se puede seguir las evoluciones más fácilmente y hay una trazabilidad de la misma. Por último, esta herramienta esta enlazada con la GitHub, la herramienta que se ha utilizado como repositorio.
- **GitHub:** [8] Repositorio donde se ha publicado el código, los dashboards... Esta herramienta es online, gratuita y solo se necesita registrarse para su utilización.
- **Overleaf:** [14] Herramienta utilizada para la realización de la memoria. Overleaf es una solución online que permite editar, trabajar en L^AT_EX,

la cual permite compartir los proyectos es gratuita y solo hace falta registrarse para poder utilizarla como en los dos casos previos.

A.2. Planificación temporal

Sobre la planificación de este trabajo se ha dividido en 5 partes (observar Figura: A.1):

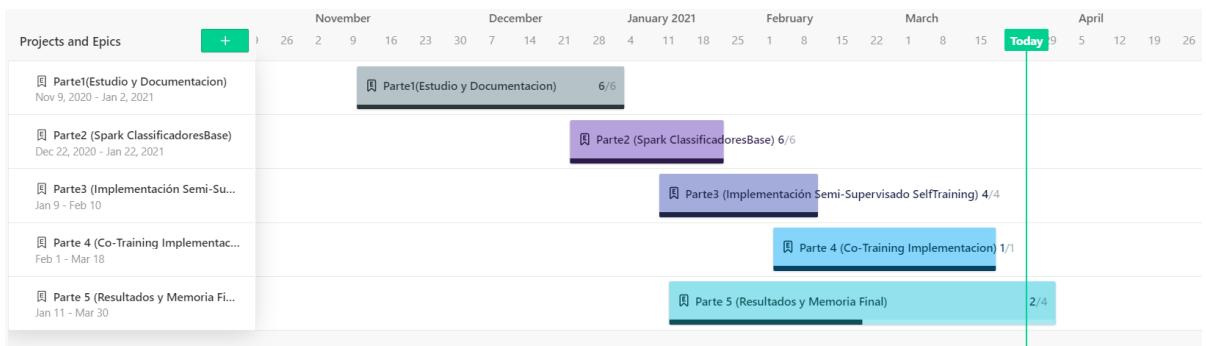


Figura A.1: Roadmap, planificación del proyecto

- 1. Estudio y documentación:** Esta primera parte más focalizada en la investigación y en entender la temática que se quería implementar, con lo cual esta muy relacionada en la lectura y búsqueda de diferentes publicaciones como se ha mencionado en la memoria y herramientas que se han utilizado.
- 2. Spark clasificadores base:** En este apartado se ha aprendido los conceptos principales en Spark ML trabajando con DataFrame y también con el concepto de Pipeline, también se han estudiado e implementado los clasificadores base (donde posteriormente se utilizaran para los algoritmos de semisupervisado). También se ha realizado la comparativa con Keel.
- 3. Implementación semisupervisado Self-Training:** Implementación y comprobación (el test como se ha explicado en 5 se ha hecho comparándolo con KEEL) del algoritmo Self-Training. También se ha realizado la comparativa con KEEL.
- 4. Implementación semisupervisado Co-Training:** Implementación y test (el test como se ha explicado en 5 se ha hecho comparándolo

con KEEL) del algoritmo Co-Training. También se ha realizado la comparativa con Keel.

5. **Resultados y Memoria Final:** Finalmente se han realizado los reports tanto con Python como con PowerBI de los resultado obtenidos con el conjunto de datos que se ha trabajado (Tabla: 5.3).

Apéndice B

Experimentación

B.1. Introducción

En este apartado se van a presentar diferentes experimentos con los conjuntos de datos que se han presentado en la Tabla 5.3 conjuntamente con los algoritmos Self-Training, Co-Training y los clasificadores base supervisados donde haremos una comparativa entre ellos. Dada la cantidad de resultados extraídos se ha creado un dashboard con Power BI el cual se puede descargar en el repositorio de Github [6].

Power BI nos ofrece la opción de poder hacer un dashboard interactivo donde nos permite navegar y extraer conclusiones de una manera rápida y fácil. En este apéndice se van a publicar los valores medios entre todos los conjuntos de datos y los más relevantes como veremos en las secciones siguientes.

Por otro lado, sobre el dashboard en Power BI se hace una explicación de cómo navegar e interpretar los resultados en el apartado E.

B.2. Objetivos generales

El objetivo de estos experimentos es entender si realmente podemos sacar ventajas trabajando con algoritmos semisupervisados (en nuestro caso para Self-Training y Co-Training) vs aprendizaje supervisado.

Para ello vamos a trabajar con datos menores a $< 20K$ y por otro lado mayores $> 1M$ más focalizados para Big Data, donde Spark y los algoritmos implementados (3, 1) tendrían más sentido. Por otro lado, los experimentos

se van a organizar como sigue. En el apartado B.3 se van a representar los resultados con los clasificadores solo supervisados y sus diferentes métricas, tal y como se ha realizado en la Figura 5.3 con la idea de entender cual es el clasificador base con mejor resultado. En el apartado B.4 se va a presentar el comportamiento de los algoritmos Self-Training y Co-Training con los diferentes thresholds y porcentaje de etiquetados utilizados.

Seguidamente en el apartado B.5 se van a comparar los resultados entre los algoritmos semisupervisados vs supervisados y se podrá ver donde se consigue mejores resultados para los casos de los semisupervisados. A continuación, en el apartado B.6 se va hacer una comparativa entre Self-Training y Co-Traininig.

Finalmente en el apartado B.8 se va a estudiar los datos etiquetados y no etiquetados al inicio y al final del proceso dependiendo del threshold seleccionado, también se va a poder ver el accuracy conseguido.

B.3. Resultados con los clasificadores de aprendizaje supervisado

En este apartado se va hacer una comparativa entre los clasificadores base (supervisados, Tabla 5.4) con diferentes métricas (accuracy, PR, AUC, F1Score) 4.3.

En la Figura B.1 se muestra el valor medio con todos los datos que aparecen en la tabla de selección. Se puede ver que RF (random forest) sería el clasificador base con mejores resultados para todas las métricas, destacar que para la mayoría de conjunto de datos sus clases están balanceadas (Tabla 5.3), con la excepción de coil2000, donde se puede ver en la Figura B.2 una clara diferencia entre la accuracy y las otras métricas en especial con el PR y el F1Score [5] , donde nos demuestra que cuando hay una alta precisión y una baja sensibilidad se puede conseguir una buena accuracy pero realmente el clasificador esta trabajando mal (métricas PR y F1Score). Para el resto de datos, al estar balanceados correctamente, no tendremos este problema.

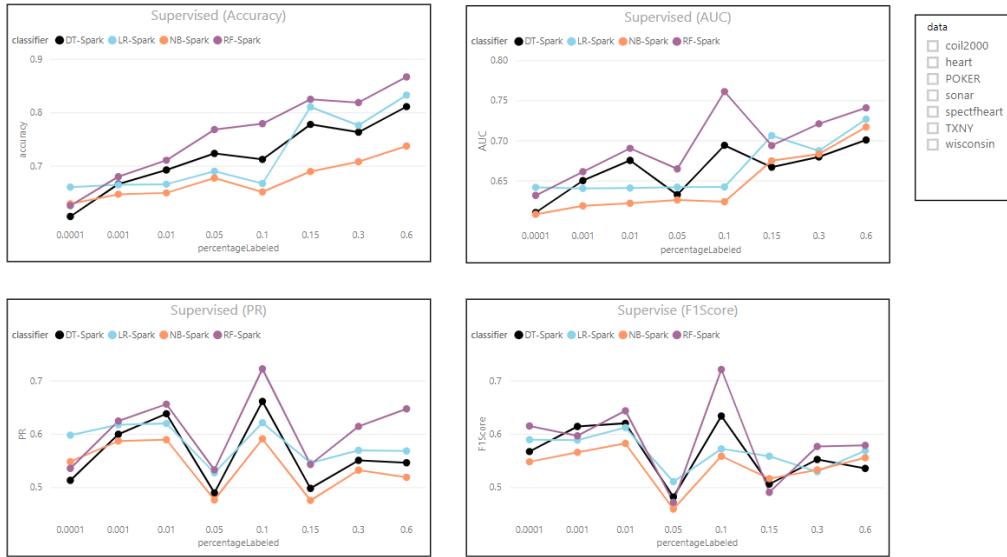


Figura B.1: Comparativa con diferentes métricas clasificadores base (supervisados). El resultado es el valor medio con todos los conjuntos de datos.

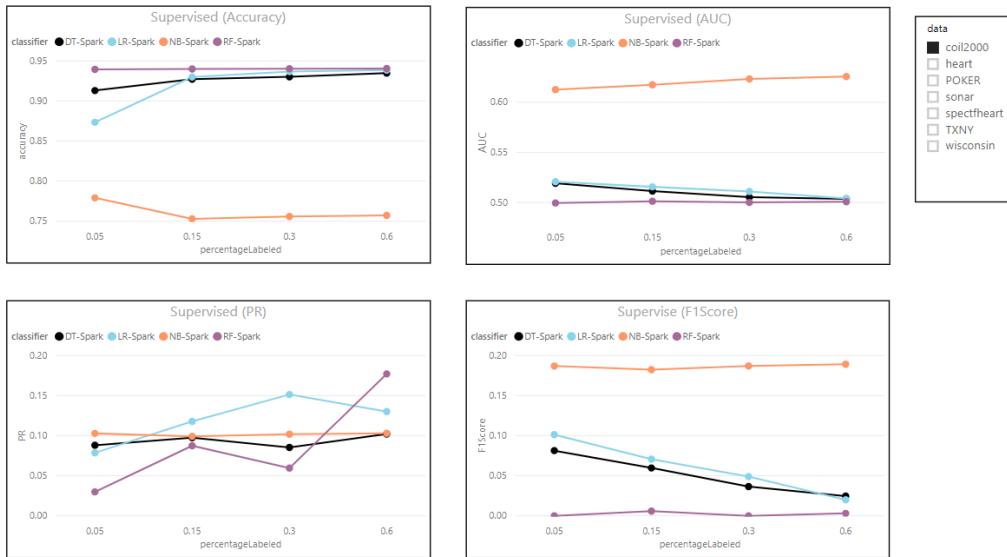


Figura B.2: Comparativa con diferentes métricas clasificadores base (supervisados). Resultado del conjunto de datos coil2000

B.4. Relación semisupervisado el threshold y el porcentaje de etiquetado seleccionado

Este apartado se va a dividir en los diferentes algoritmos semisupervisados (Self-Training, Co-Training) en función del tamaño del conjunto de datos, uno para datos menores de 20k y el otro para datos mayores de 1M de instancias.

Relación Self-Training y el threshold seleccionado datos < 20k instancias

En la Figura B.3 se puede ver el efecto entre el threshold seleccionado, el porcentaje etiquetado ¹ y el clasificador base utilizado.

Se ve como los mejores resultados se obtienen con RF como clasificador base, con un threshold de 0,9 y un porcentaje de etiquetado del 60 %. Aunque se puede apreciar que la diferencia con el 30 % no es tan dispar, eso nos indicaría que trabajando con un 30 % de porcentaje etiquetado se conseguirían resultados similares y con menos recursos para generar el modelo.

En la Figura B.4 se puede apreciar mejor la tendencia para Random Forest (RF) donde el resultado sería el valor medio del accuracy con todos los datos menores a 20k instancias. Se ve que hay una cierta mejora cuanto más crece el threshold.

Por otro lado, en la Figura B.5 se puede ver que se ha modificado el eje x donde se ha introducido el porcentaje de etiquetado, de esta manera se puede ver la tendencia en función de este. También se puede apreciar que no hay una mejoría destacable cuando este es mayor de 0,3 (30 %).

Relación entre Self-Training y el threshold seleccionando datos > 1M de instancias

Siguiendo el mismo criterio que el apartado anterior para los datos menores de 20k instancias, se hará una comparativa con los datos mayores de 1M de instancias. En la Figura B.6 se puede apreciar una tendencia similar a la Figura B.3 pero para este caso con los datos de 1M.

¹Remarcar que el porcentaje que utilizamos es siempre tanto por 1.

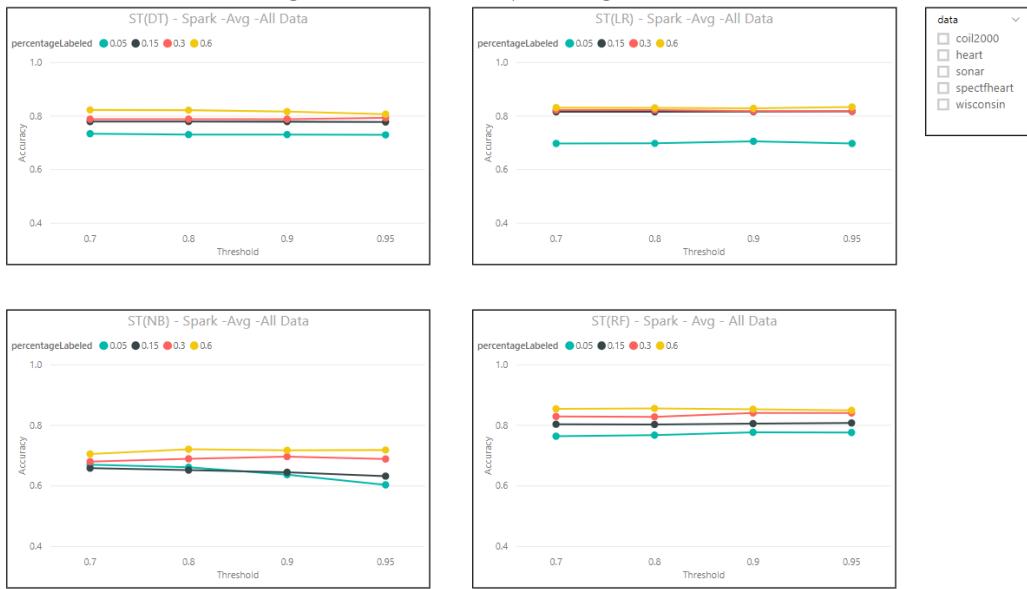


Figura B.3: Valor medio accuracy con todos los conjuntos de datos menores a 20k. Resultado entre threshold y porcentaje etiquetado Self-Training

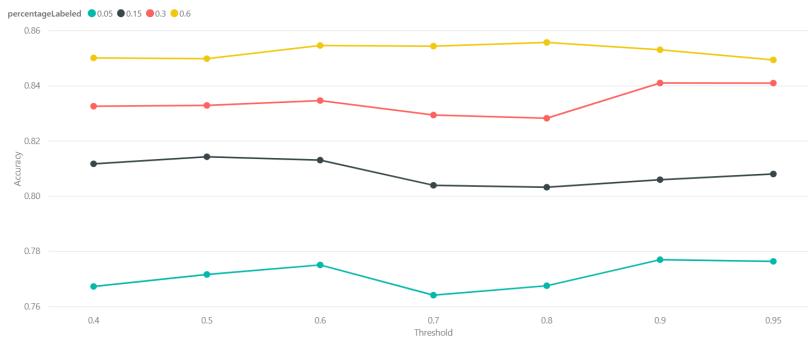


Figura B.4: Valor medio accuracy con todos los conjuntos de datos menores a 20k de instancias con Random Forest (RF). Resultado entre threshold y porcentaje etiquetado Self-Training

Por otro lado, en la Figura B.7 se muestra la misma relación pero se ha cambiado el eje x por el porcentaje etiquetado.

Se aprecia que para el Self-Training con Naive Bayes (NB) y Logistic regression (LR) como clasificador base prácticamente no hay diferencia en el resultado con diferentes datos etiquetados. Por otro lado, Decision Tree (DT) tiene una cierta mejoría hasta llegar a 0,05 (5 %) de porcentaje

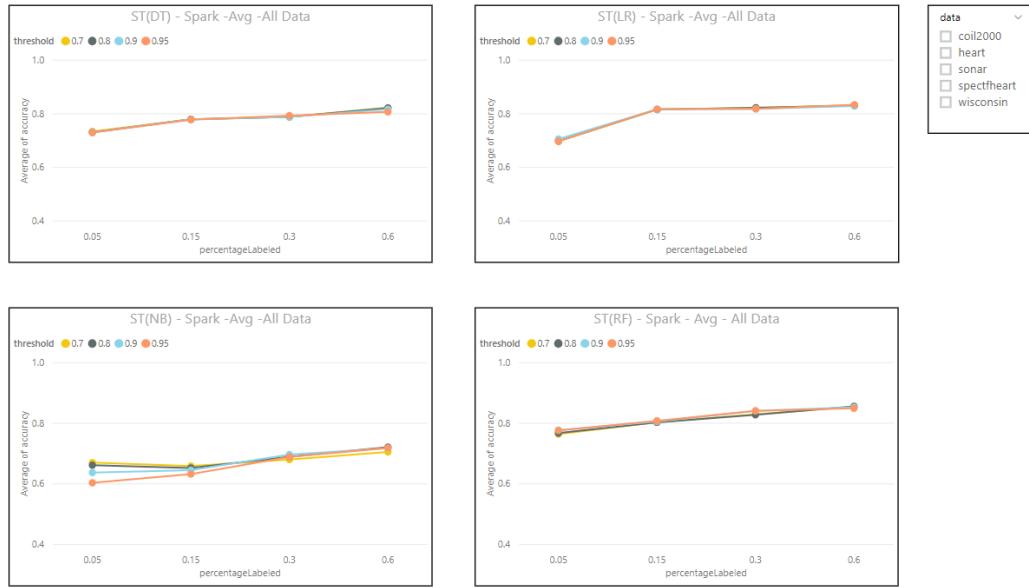


Figura B.5: Valor medio accuracy con todos los conjuntos de datos menores a 20k de instancias. Resultado entre el porcentaje etiquetado y el threshold Self-Training

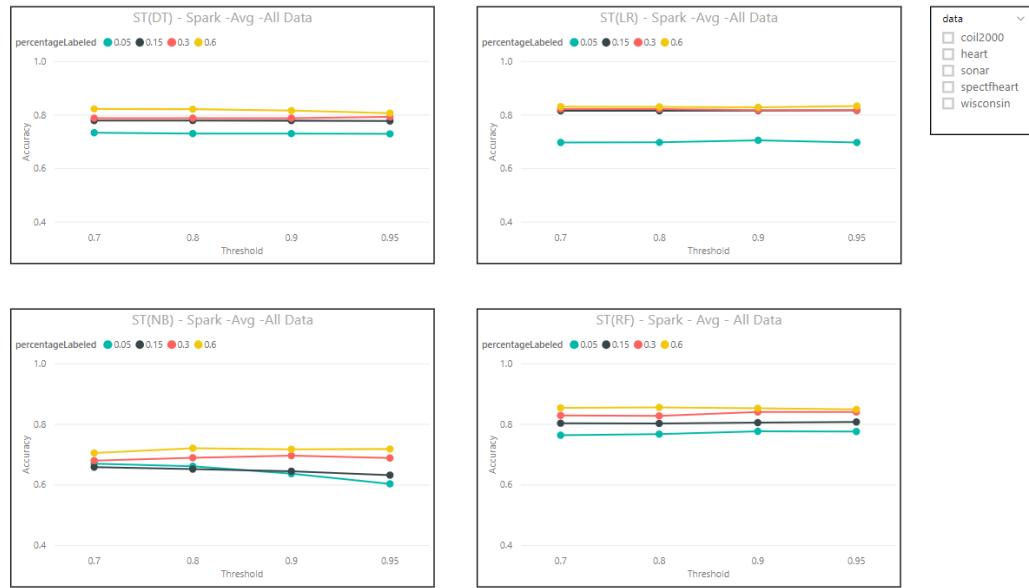


Figura B.6: Valor medio accuracy con todo los conjunto de datos mayores a 1M. Resultado entre threshold y porcentaje etiquetado Self-Training

etiquetado, por último vemos que Random Forest (RF) tiene nuevamente el mejor rendimiento el cual deja de mejorar cuando llega al 0,1 (10 %) de datos etiquetados. Si ampliamos la gráfica de RF (Figura: B.8) observamos que se obtiene el mejor resultado con un threshold de 0.9 (aproximadamente) para cualquier porcentaje de datos etiquetados.

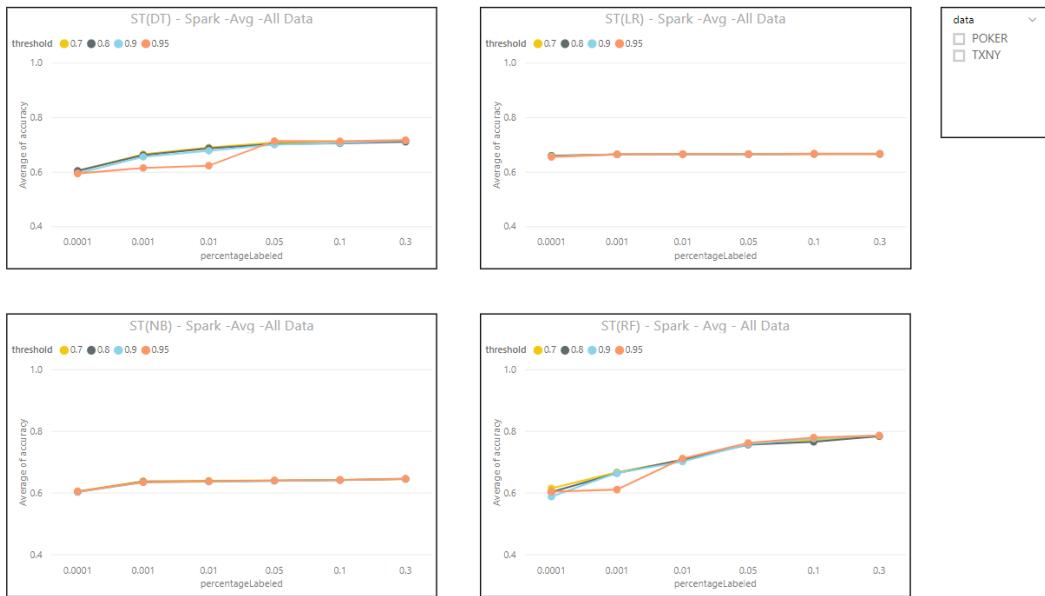


Figura B.7: Valor medio accuracy con todos los conjuntos de datos menores a 1M de instancias. Resultado entre el porcentaje etiquetado y el threshold Self-Training

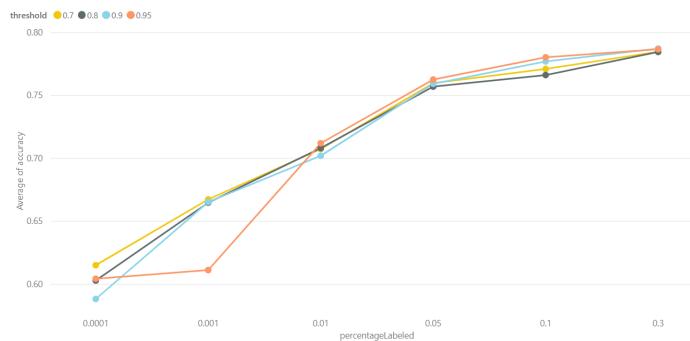


Figura B.8: Valor medio accuracy con todo los conjunto de datos Mayores a 1M de instancias con Random Forest (RF). Resultado entre threshold y porcentaje etiquetado Self-Training

Relación Co-Training y el threshold seleccionado datos < 20k instancias

En el siguiente apartado vamos a ver el comportamiento con el threshold y el porcentaje de datos etiquetados para Co-Training con los datos menores de 20K instancias.

En la Figura B.9 vemos que la tendencia es similar que para Self-Training pero hay menos diferencias en la accuracy entre el threshold seleccionado y el porcentaje etiquetado. En el apartado B.6 se va hacer una comparativa de resultado entre Self-Training y Co-Training en más detalle.

Por otro lado también se puede ver que los mejores resultados se obtienen con Random Forest (RF) como clasificador base.

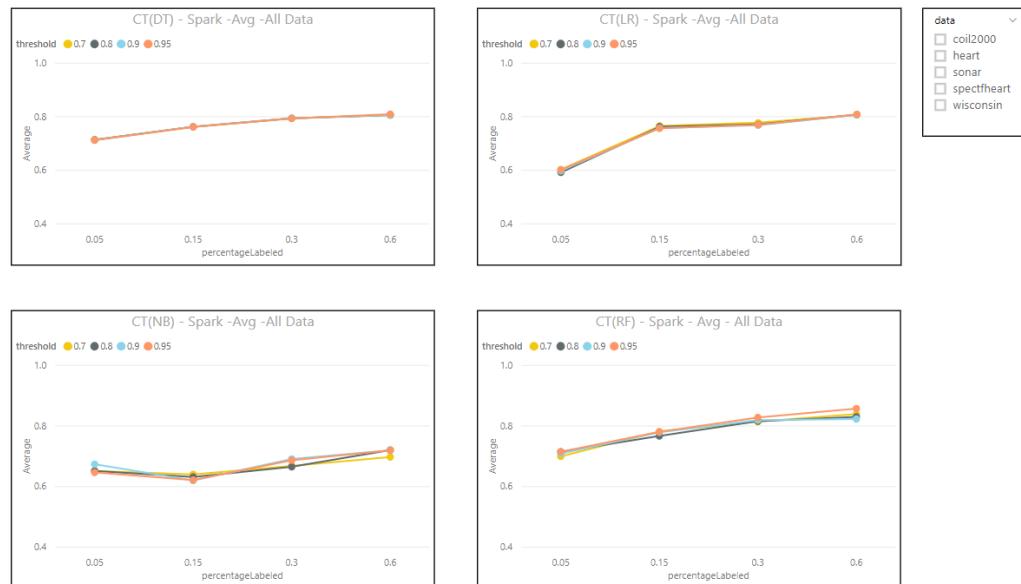


Figura B.9: Valor medio accuracy con todo los conjunto de datos menores a 20k de instancias. Resultado entre el porcentaje etiquetado y el threshold Co-Training

Relación Co-Training y el threshold seleccionado datos > 1M de instancias

Manteniendo el mismo criterio que para el Self-Training en este apartado vamos a ver los resultados obtenidos con Co-Training para diferentes porcentajes de etiquetado y con diferentes thresholds.

En la Figura B.10 vemos una tendencia muy similar entre Self-Training (Figura B.7) donde el mejor resultado se consigue con RF y también se mantiene la misma tendencia para los Co-Training con NB y LR.

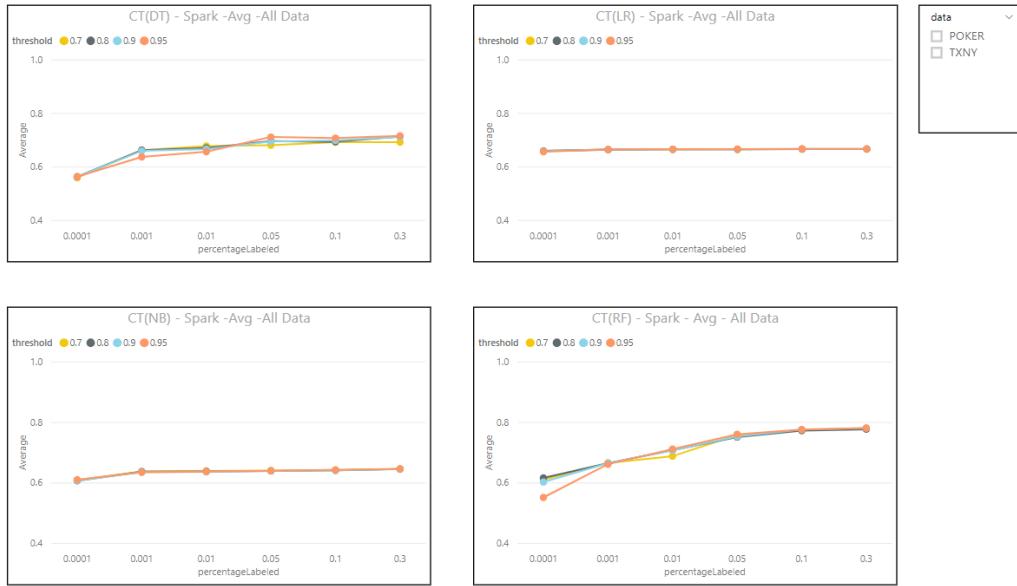


Figura B.10: Valor medio accuracy con todo los conjunto de datos mayores a 1M de instancias. Resultado entre el porcentaje etiquetado y el threshold Co-Training

Con los resultados obtenidos hasta ahora se podría decir que tanto Self-Training como Co-Training tendrían mejores resultados con un threshold de 0.9 y con un clasificador base RF.

B.5. Comparativa semisupervisado vs supervisado

A continuación se van a comparar los algoritmos Self-Training y Co-Training con los clasificadores base trabajando de forma independiente. Siguiendo el mismo criterio, se van a separar por algoritmo y por número de instancias(< 20k instancias y >1M de instancias).

Finalmente se ha seleccionado un threshold de 0,9 para todo los casos, dado que es uno de los threshold donde se ha conseguido mejores resultados (sección B.4).

Comparativa Self-Training vs supervisado para datos < 20k instancias

En la Figura B.11 se puede observar que aparece una mejora por parte de Self-Training con respecto al clasificador supervisado para los casos de DT, LR, RF con un porcentaje de etiquetado de 0.15 (15 %), 0.15 (15 %) y 0.3 (30 %) respectivamente donde la mejora en la accuracy es de 0.1 (1 %).

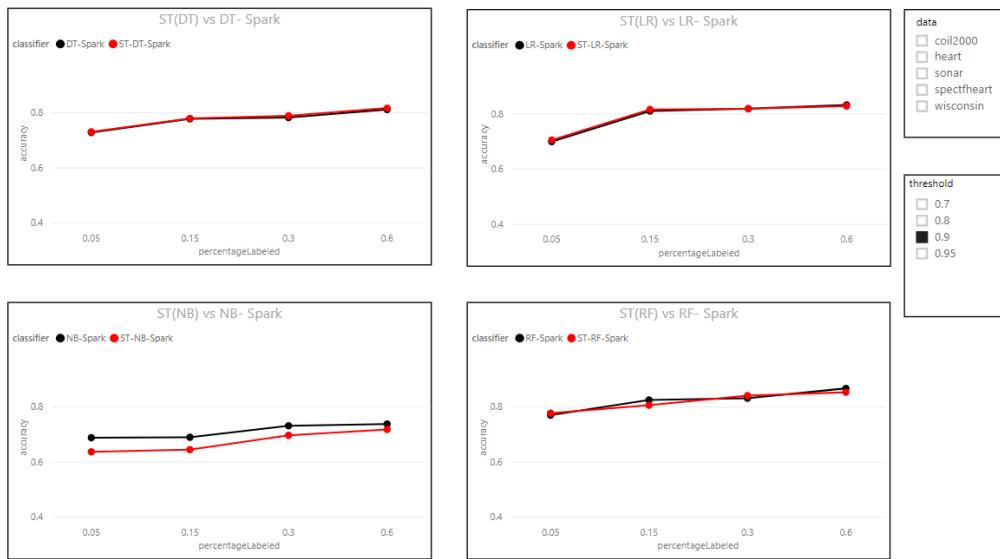


Figura B.11: Valor medio accuracy y porcentaje etiquetado para Self-Training y clasificadores Supervisado con todo los conjunto de datos menores a 20k instancias. Comparativa entre Self-Training vs Supervisado

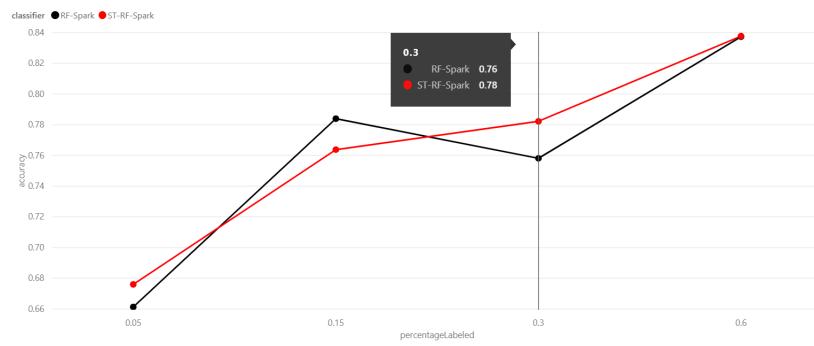


Figura B.12: Accuracy y porcentaje etiquetado para Self-Training y RF con el conjunto de datos heart. Comparativa entre Self-Training vs Supervisado

Por otro lado, si observamos (Figuras B.12 y B.13) el comportamiento de Self-Training trabajando con RF se consiguen mejores resultados que utilizando RF trabajando independientemente para los conjuntos de datos heart y sonar todos ellos con un 0,3 (30 %) de porcentaje etiquetado.

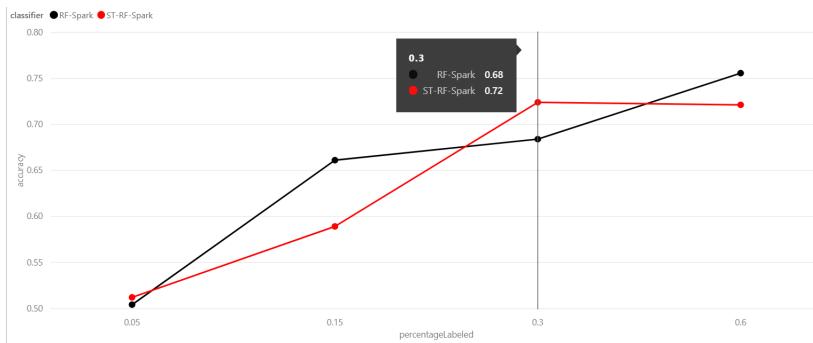


Figura B.13: Accuracy y porcentaje etiquetado para Self-Training y RF con el conjunto de datos sonar. Comparativa entre Self-Training vs Supervisado

Comparativa Self-Training vs supervisado para datos > 1M de instancias

Por otro lado, con el criterio que se ha ido siguiendo hasta ahora vamos a analizar los resultados de Self-Training con los clasificadores supervisados para los datos menores a 1M de instancias. Como se ha hecho previamente se ha seleccionado un threshold de 0,9.

En la Figura B.14 se puede apreciar que no hay una mejora significativa trabajando con Self-Training en vez de con clasificadores supervisados para el conjunto de datos utilizados. Esto no quiere decir que siempre siga el mismo patrón para un número de instancias similar.

Comparativa Co-Training vs supervisado para datos < 20k instancias

Para el caso de el algoritmo semisupervisado Co-Training se ha comparado de la misma forma que para Self-Training.

En la Figura B.15 se hace una comparativa con los resultados de accuracy medios trabajando con todos los datos menores de 20k. En general se puede apreciar que no hay una diferencia significativa entre ambos, pero para los casos de Co-Training trabajando con RF como clasificador base se consigue

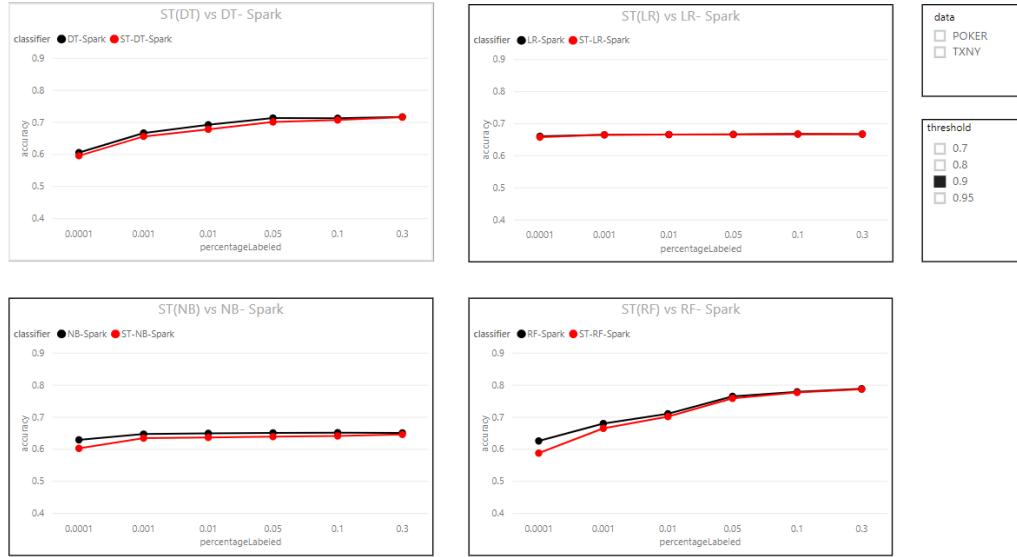


Figura B.14: Valor medio accuracy y porcentaje etiquetado para Self-Training y clasificadores Supervisado con todo los conjunto de datos mayores a 1M de instancias. Comparativa entre Self-Training vs Supervisado

un mejor resultado que el clasificador RF para un porcentaje de etiquetado de 0,3 (30 %) los mismo para el caso de Co-Training con DT como clasificador base respecto a trabajar con DT supervisado (independientemente).

Si hacemos un análisis más detallado por conjunto de datos, observamos que para el conjunto heart obtenemos una mejora con Co-Training trabajando con DT, RF, LR como clasificador base respecto a los clasificadores supervisados. Para ello observar la Figura B.16 remarcado en rojo. Donde la mejoría en la accuracy es entre el 0,01 (1 %) y el 0,05 (5 %).

Por otro lado, la Figura B.17 representa los resultados del conjunto de datos sonar los cuales consiguen unos mejores resultados con Co-Training para RF y DT como clasificador base respecto a los clasificadores RF y DT trabajando como supervisado. Para el caso de RF repite la mejora con un 0,3 (30 %) de datos etiquetados.

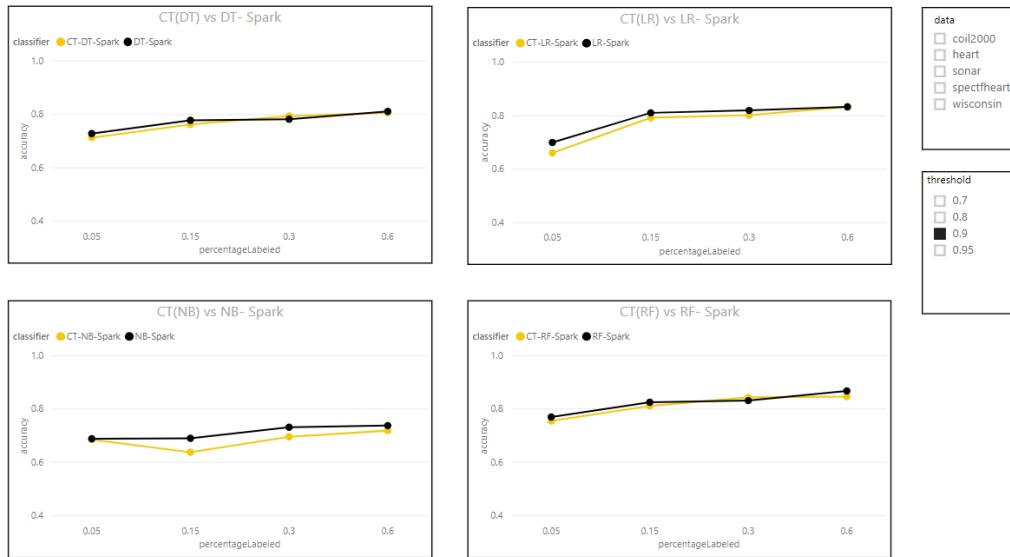


Figura B.15: Valor medio accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con todo los conjunto de datos menores a 20k de instancias. Comparativa entre Self-Training vs Supervisado

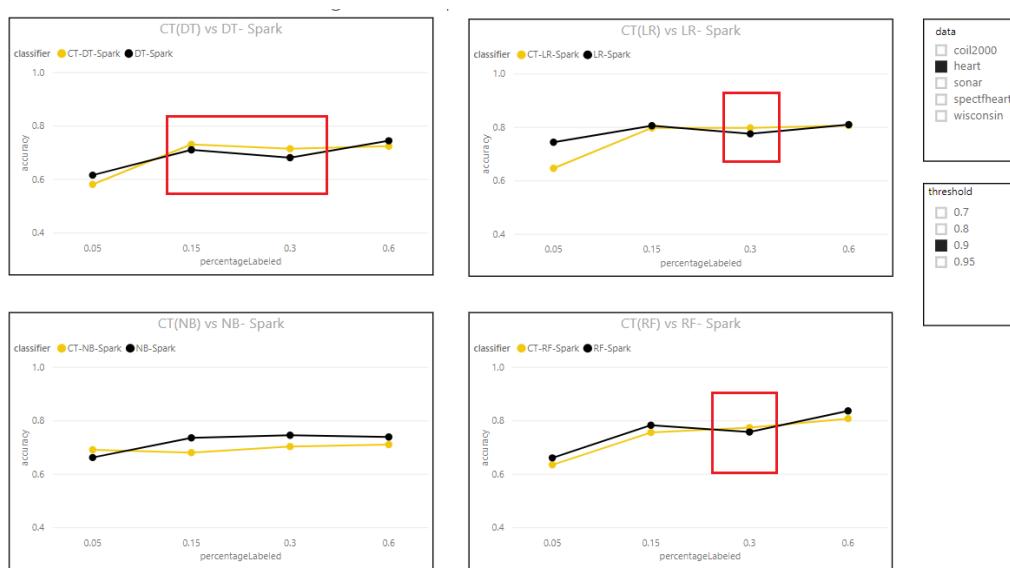


Figura B.16: Accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con el conjunto heart. Comparativa entre Self-Training vs Supervisado

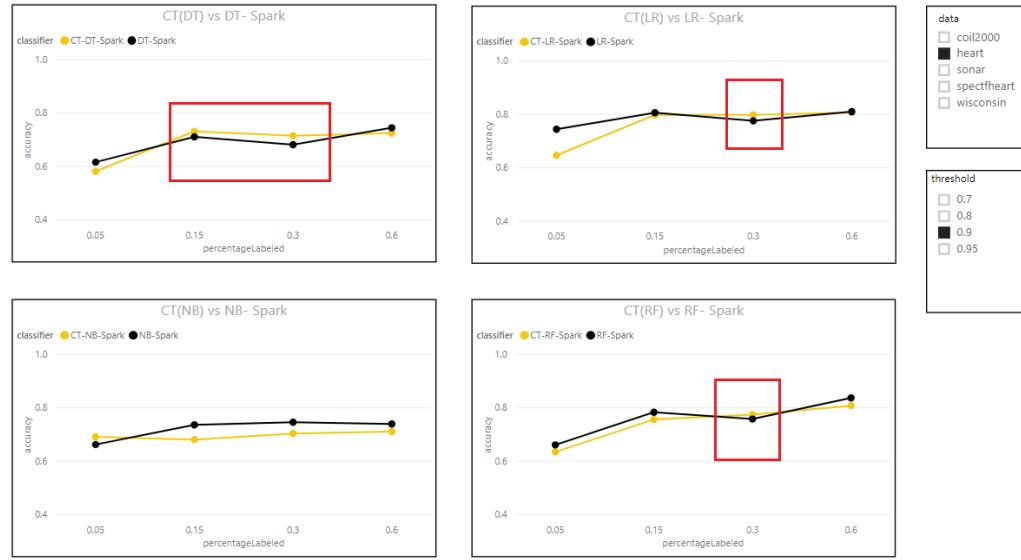


Figura B.17: Accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con el conjunto sonar. Comparativa entre Self-Training vs Supervisado

Comparativa Co-Training vs supervisado para datos > 1M de instancias

Para el caso de Co-Training con conjunto de datos mayores a 1M de instancias, se observa en la Figura B.18 que la tendencia es similar que la Figura B.14 para el caso de Self-Training.

B.6. Comparativa Self-Training vs Co-Training

En este apartado se van a comparar los dos algoritmos semisupervisado diseñados. Para ello vamos a mantener el mismo criterio y vamos a dividir los resultados en grupos de datos (menores de 20k y mayores de 1M de instancias) y se va a trabajar con un threshold de 0,9.

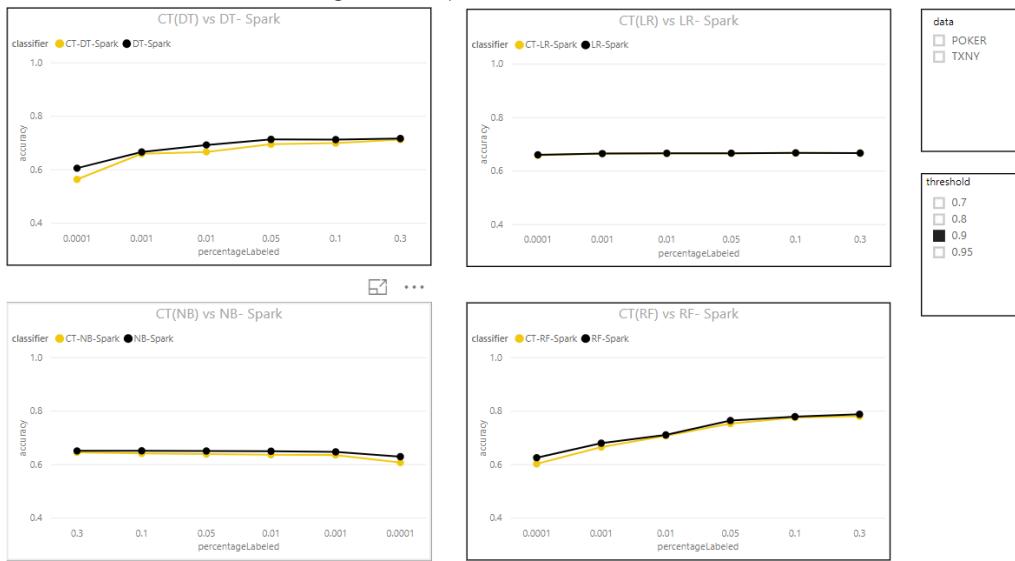


Figura B.18: Valor medio accuracy y porcentaje etiquetado para Co-Training y clasificadores Supervisado con todo los conjunto de datos mayores a 1M de instancias. Comparativa entre Self-Training vs Supervisado

Comparativa Self-Training vs Co-Training con datos < 20k instancias

En la Figura B.19 se muestran los resultados medios en accuracy de Self-Trainig y Co-Training para los datos menores a 20k, se puede apreciar que no hay una gran diferencia entre ellos pero si que se puede observar una ligera mejoría para los casos de Self-Training.

Si nos focalizamos en algún caso concreto de datos. En este caso para sonar y con RF como clasificador base, se puede ver (Figura B.20) que si que conseguiría una mejora en Co-Training respecto Self-Traininig. Esto indicaría que no se puede seguir al pie de la letra el patrón que indican los resultados medios y que para cada conjunto de datos esto podría variar.

Comparativa Self-Training vs Co-Training con datos > 1M de instancias

Por último hacemos la comparativa para Self-Training y Co-Training para los conjuntos de datos mayores a 1M de instancias.

En la Figura B.21 se observa el valor medio en la accuracy para los conjuntos de datos mayores a 1M de instancias. Para entender más sus

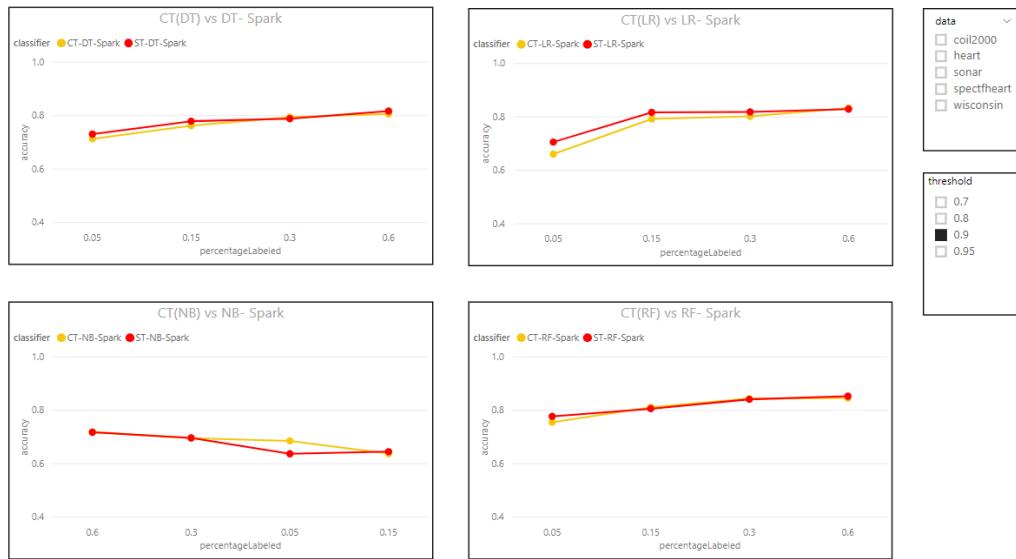


Figura B.19: Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con todo los conjunto de datos menores a 20k instancias. Comparativa entre Self-Training vs Co-Training

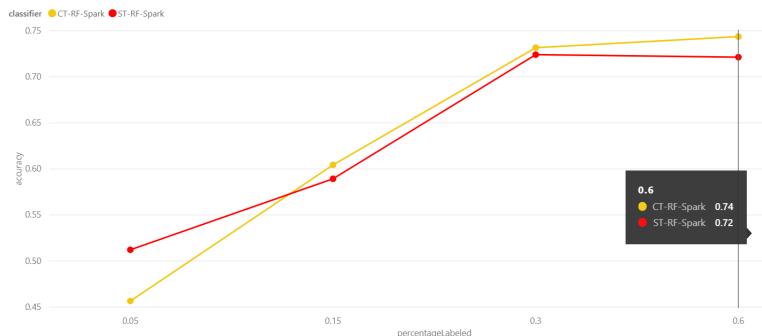


Figura B.20: Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training el conjunto de datos sonar con RF. Comparativa entre Self-Training vs Co-Training

diferencias seleccionamos y analizamos los conjuntos de datos por separado y para ello también seleccionamos los casos con que los algoritmos Self-Training y Co-Training trabajan con RF como clasificador base, ya que para RF es donde se consiguen en general mejores resultados.

Para el caso de Poker (Figura B.22) se aprecia que Self-Training tendría un mejor funcionamiento.

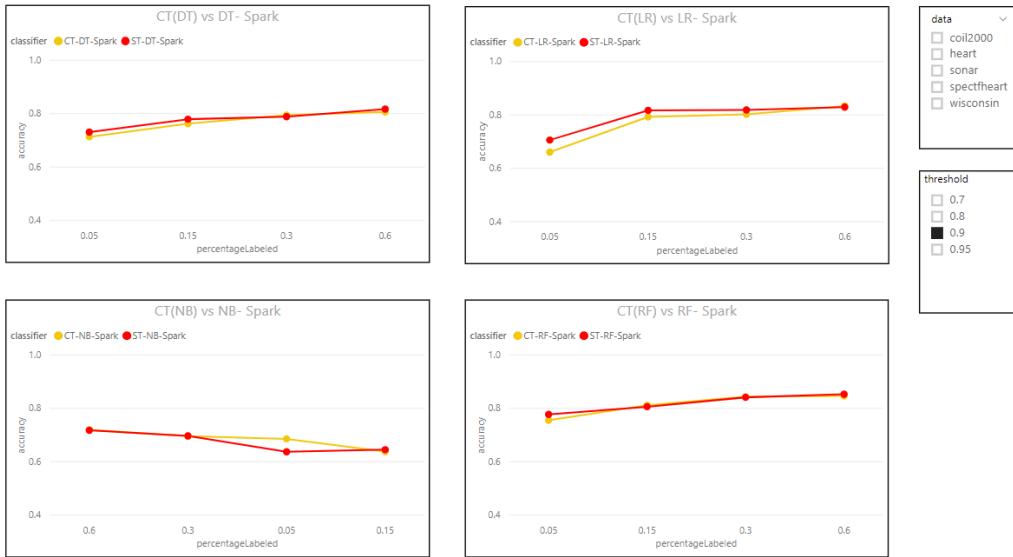


Figura B.21: Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con todos los conjunto de datos menores a 20k instancias. Comparativa entre Self-Training vs Co-Training

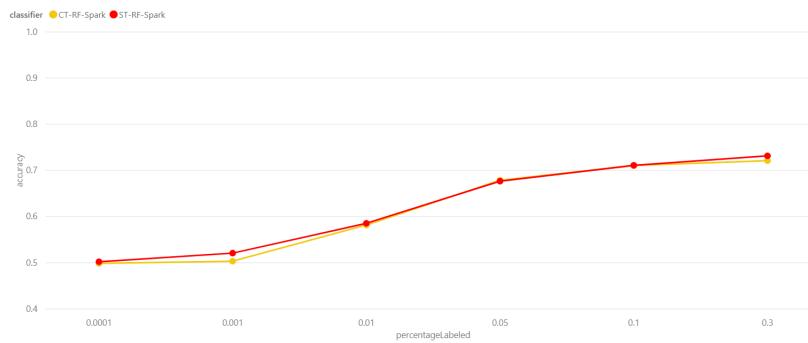


Figura B.22: Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con el conjunto de datos Poker. Comparativa entre Self-Training vs Co-Training

En cuanto al otro conjunto de datos TXNY, se puede ver en la Figura B.23 que para este caso se consigue un mejor resultado con Co-Training donde se consigue una mejora en accuracy de 0,04 (4%).

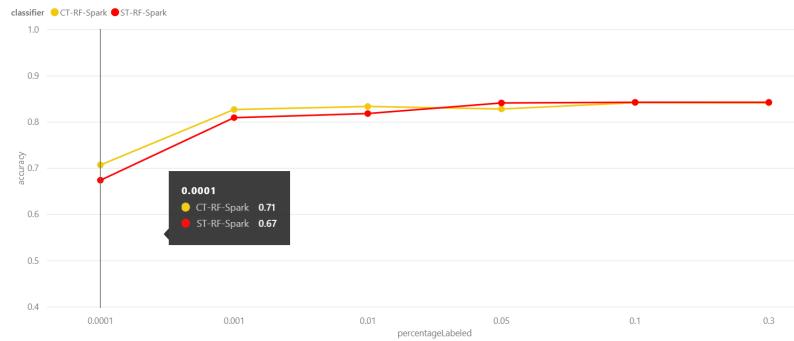


Figura B.23: Valor medio accuracy y porcentaje etiquetado para Co-Training y Self-Training con el conjunto de datos TXNY. Comparativa entre Self-Training vs Co-Training

B.7. Conclusiones de los experimentos

Como se ha podido ver en los apartados anteriores, para algunos casos concretos se puede obtener una mejoría trabajando con algoritmos semi-supervisados. Los mejores casos se consiguen con un threshold de 0.9 y trabajando con RF como clasificador base.

En cuanto si es mejor trabajar con Self-Training o Co-Training, en términos generales se podría decir que Self-Training es mejor candidato dado que es un algoritmo que necesita menos recursos y en muchos de los casos se consiguen resultados similares aunque dependerá del conjunto de datos.

Hay que tener en cuenta que no se consiguen grandes mejoras si las expectativas son de mejorar el accuracy en mas de un 10 % pero sí que es cierto que en algunos casos se ha llegado a mejorar más del 5 % el accuracy.

Dado a la complejidad de los datos y el tiempo para este proyecto sería interesante en el futuro buscar patrones que nos indiquen para qué conjuntos de datos (debido a su casuística) puede ser mejor trabajar con semisupervisado o supervisado, dado que con los resultados en los experimentos ejecutados en este proyecto se aprecia que el semisupervisado funcionaría mejor que supervisado dependiendo del tipo de conjunto de datos y del porcentaje de clases etiquetadas. Este concepto es conocido como meta-learning [34].

B.8. Relación entre los datos Etiquetados (inicio del y final)

Esta sección pretende presentar otra visualización que se ha implementado en Power BI para poder ver la relación entre los datos etiquetados al inicio y al final del proceso en función de sus datos etiquetados. Dado el número de combinaciones hemos seleccionado trabajar con un Threshold de 0.9 y con ST-RF-Spark.

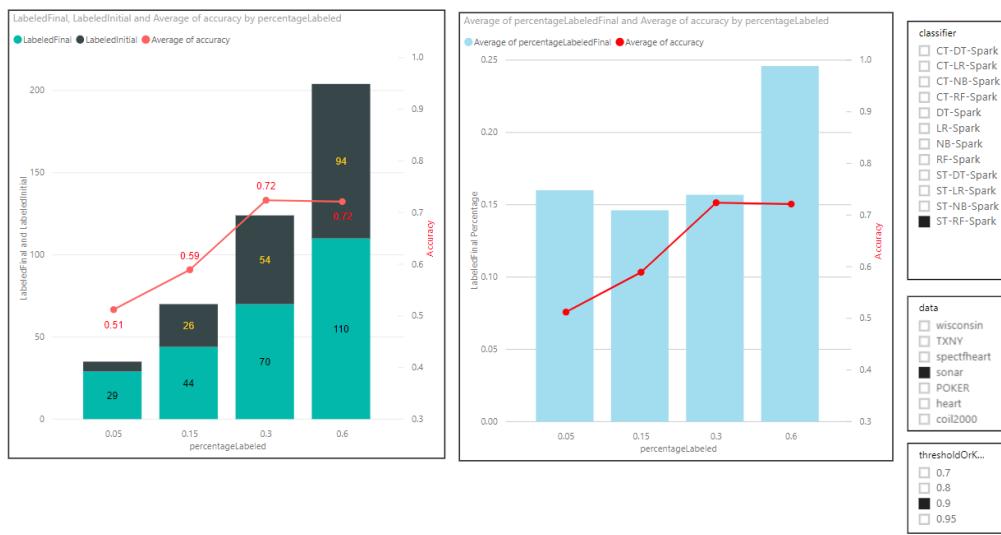


Figura B.24: Relación entre el etiquetado inicial, final y el porcentaje etiquetado

En la Figura B.24 se puede ver como a partir de cierto porcentaje de etiquetado la accuracy se estabiliza y no se consiguen mejoras. Este es un hecho común entre los conjuntos de datos *TXNY*, *sonar*, *POKER*, con lo cual hay que tener en cuenta cual es el porcentaje de etiquetado más eficiente por conjunto de datos ya que si se selecciona un porcentaje muy elevado es posible que se incremente el coste computacional y no se consigan mejoras significativas.

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección se va a introducir el tipo de diseño utilizado para la implementación de los algoritmos Self-Training y Co-Training con Spark basado en el concepto de pipeline (ver Figura: 4.4, apartado: 4.2).

C.2. Diseño en Spark

Para el diseño de este trabajo y la posterior realización de los experimentos (apartado: B) hemos diseñado una serie de clases y objetos (todos ellos definidos en el apartado: D.3) totalmente parametrizables para su fácil implementación y modificación sobretodo para utilizarlo en un entorno de experimentación e investigación. Para ello hemos utilizado el concepto de Pipeline que integra la librería de Spark ML (Cabe remarcar que no se puede trabajar con pipeline con Spark MLlib), la principal idea como su nombre indica es que trabaja con el concepto de “tubería“ donde se le pueden unir tantos módulos de estimadores como transformadores se necesiten, lo cual da mucha flexibilidad al diseño, ya que para posibles modificaciones simplemente se tendría que cambiar un módulo por otro, añadir o eliminar el que se deseé. Estos módulos, Spark los define con una clase llamada PipelineStage ([2]).

En la Figura C.1 se ha presentado el pipeline utilizado para este proyecto, en este caso tendríamos tres pipelines, uno para cada clase de aprendizaje, **Supervisado** (Supervised), el siguiente para **Self-Training** y por último el último para **Co-Training**. La primera parte del los pipelines es común seria

el proceso de **Featurization** el cual transformaría el DataFrame de entrada (*Datos de entrenamiento, línea verde y Datos de test, linea azul. Figura: C.1*) acorde a como trabajan las clases de Spark ML, este sería un DataFrame con un vector de features (atributos) y labels (clases) en formato numérico. Despúes del proceso de Featurizacion, desetiquetaríamos parte de los atributos de entrenamiento en función del porcentaje seleccionado con (o módulo en el pipeline) **UnLabeledTransformer** y posteriormente entrenariámos con sus datos de entrenamiento cada uno de los modelos (**Supervised**, **Self-Training** y **Co-Training**) donde generariámos un Transformer (o modelo) el cual lo vamos a utilizar para predecir las clases de los datos de test. Por último ya fuera del pipeline, pero igualmente importante (ya que forma parte del diseño y de los experimentos), una vez obtenemos las predicciones finales de cada modelo utilizaremos el objeto **FunctionsSemiSupervised**, concretamente la función **supervisedAndSemiSupervisedResults** (observar la tabla: D.6) donde añadimos los resultados en un DataFrame de resultados donde posteriormente generaremos el reporte en Python y el dashboard en PowerBI.

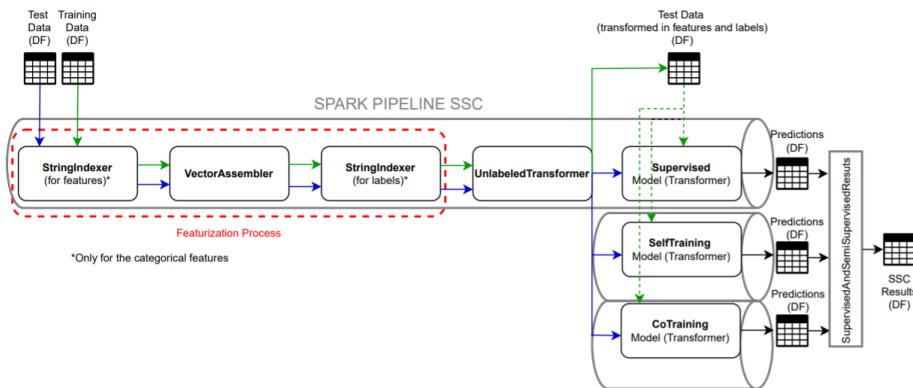


Figura C.1: Pipeline utilizado para el diseño del proyecto

C.3. Diseño arquitectónico

En relación a la arquitectura, como se ha indicado en el apartado 4 vamos a trabajar principalmente con Azure como infraestructura y dentro de la misma con DataBricks como una solución PaaS (Platform as a Service). En la Figura C.2 se representa cómo se va a trabajar y conectar Databricks con los datos en bruto (raw) y una vez se obtengan los resultados, como

estos se van a conectar con las herramientas que se van a utilizar para su visualización (Python y PowerBI)

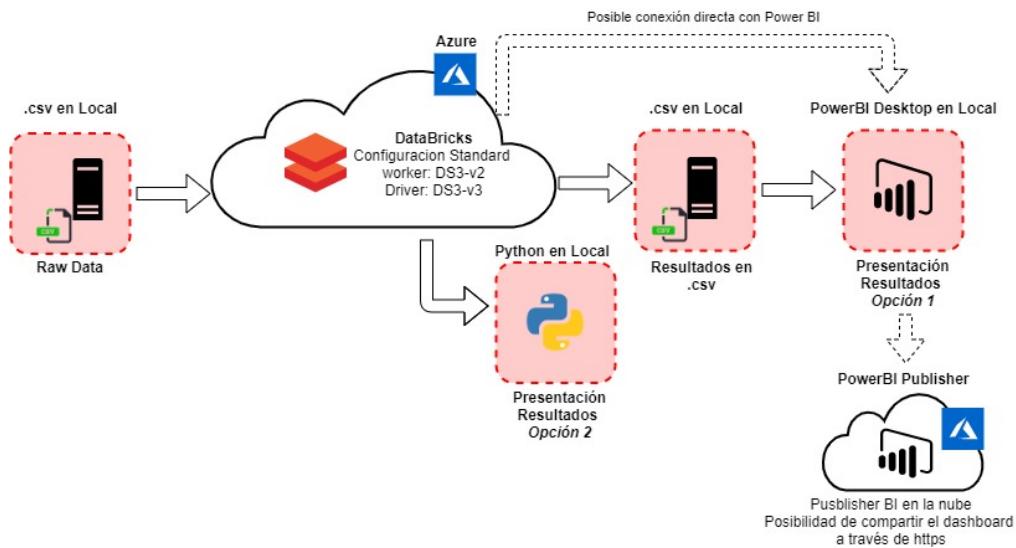


Figura C.2: Arquitectura y diseño utilizado

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se va hacer referencia a la estructura del código del proyecto focalizándonos en las clases y objetos creados, cabe remarcar que las clases creadas para cada uno de los métodos semisupervisados es totalmente parametrizable y adaptable a la arquitectura de Spark como se ha mencionado previamente.

D.2. Versiones utilizadas para DataBricks, Spark, Scala, Python y Power BI

En este proyecto se ha utilizado, **DataBricks** con la versión *7.4*, **Spark** con la versión *3.0.1*, **Scala** con la versión *2.12*. Remarcar que al trabajar con DataBricks en Azure como una solución PaaS (Platform as a Services) siempre tenemos la ultima versión instalada sin necesidad de hacer patching manualmente.

Por otro lado la versión utilizada en **Python** es la *versión 3*, finalmente en relación a la versión de **Power BI Desktop** esta es la *2.65.5313.701 64-bit*.

D.3. Clases y Objetos

En este apartado se va a definir cada una de las características (métodos, extensión, parámetros...) de las clases y las funciones(estas últimas encapsuladas en un *object*)

Clase SemiSupervisedDataResults

Clase	Descripción de clases	Extensión	Parámetros	Métodos	Descripción
SemiSupervisedDataResults	Resultados generados por los algoritmos semisupervisados: dataLabeledFinal, dataLabeledInicial, dataUnlabeledInicial...	No aplicable	No aplicable	No aplicable	No aplicable

Tabla D.1: definición de la clase SemiSupervisedDataResults

Clase UnlabeledTransformer

Esta clase, Tabla D.2 se va a encargar de hacer la partición de los datos entre etiquetados y no etiquetados, con lo cual el principal objetivo de esta clase es des-etiquetar las clases de los diferentes conjunto de datos.

Clase	Descripción de clases	Extensión	Parámetros	Métodos	Descripción
UnlabeledTransformer	Clase encargada de hacer la partición de los datos etiquetados (Labeled) y no etiquetados (Unlabeled) en este caso serán etiquetados como NaN	Transformer	No Aplicable	setPercentage	Introducción del porcentaje etiquetado sobre el conjunto de datos
				setColumnName	El nombre de la columna donde estaran los datos etiquetados y no etiquetados
				setSeed	Elección de la semilla para la partición
				transform	Nos devuelve la columna seleccionada con los nuevos datos (Etiquetados y no Etiquetados)

Tabla D.2: definición de la clase UnlabeledTransformer

Clase Supervised

Una vez se tienen datos no etiquetados y etiquetados esta clase los identifica y selecciona solo los etiquetados para trabajar con los clasificadores base seleccionados (introducidos como parámetros). Tabla D.3

Clase	Descripción de clases	Extensión	Parámetros	Métodos	Descripción
Supervised	Encargada de seleccionar los datos etiquetados y no etiquetados en este caso, para supervisado solo se utilizaran los datos etiquetados	ProbabilisticClassifier	ClasificadorBase ejemplo: NB,DT, LR...	setColumnLabelName	Selección de la columna donde estan las clases etiquetadas y no etiquetadas
				fit/train	Entrenan el modelo supervisado solo con los datos etiquetados seleccionados previamente
				transform	Devuelve un DataFrame con las predicciones utilizando el modelos previamente entrenado

Tabla D.3: definición de la clase Supervised

Clase Self-Training

La clase Self-Trainig es la encargada de ejecutar el algoritmos Self-Training, en la Tabla D.4 se especifican todos sus métodos y como se utiliza la misma.

Clase Co-Training

La clase CoTrainig es la encargada de ejecutar el algoritmos Self-Training, en la Tabla D.5 se especifican todos sus métodos y como se utiliza la misma.

Objeto functionSemiSupervised

En esto objeto (Tabla D.6) están definidas las funciones utilizadas para el diseño donde creamos los pipelines para cada uno de los algoritmos utilizados (Self-Training ,Co-Training) y donde se calculan los resultados con sus métricas y se genera el DataFrame con todos los resultados.

D.4. Creación de las librerías para los algoritmos semisupervisados

En esta apartado se va a describir como vamos a crear un paquete .jar con las clases explicadas en el apartado D.3. Una vez creado el .jar se tendrá que añadir en DataBricks (también se puede añadir a otras plataformas no es solo exclusivo de DataBricks).

La Figura D.1 muestra los diferentes pasos para poder crear el paquete que posteriormente utilizaremos para importar las librerías (*import org.apache.spark.ml.semisupervised...*). Antes de todo tendremos que instalar

Clase	Descripción de clases	Extensión	Parámetros	Métodos	Descripción
Self-Training	Clase donde se ha implementado el algoritmo de SelfTraining la cual entrena con .fit el modelo y devuelve las predicciones de las clases con transform	ProbabilisticClassifier	ClasificadorBase ejemplo: NB, DT, LR...	setSemiSupervisedDataResults setColumnLabelName setThreshold setCriterion setMaxIter setKbest getDataLabeledFinal getUnDataLabeledFinal getDataLabeledIni getUnDataLabeledIni getIter fit/train transform	Pasamos por parámetro la clase SemiSupervisedDataResults donde tendremos los resultados relacionados con los algoritmos semisupervisados Selección de la columna donde están las clases etiquetadas y no etiquetadas Selección del umbral de probabilidad donde se indicara si una instancia no etiquetada debe ser actualizada en el conjunto de datos etiquetados valores entre 0 y 1, se necesita seleccionar "Threshold" en setCriterion Selecciona si se quiere trabajar con threshold (umbral) o de lo contrario se decide un porcentaje de datos a etiquetar (kBest), este es un valor String, "Threshold." o "kBest" Número de iteraciones máximas según el algoritmo Self-Training, valores enteros Porcentaje de valores No etiquetados que se desean etiquetar cuando se ha seleccionado "kBest" en SetCriterion Obtención del número de datos etiquetados una vez finalizado el método Self-Training Obtención del número de datos no etiqueta dos una vez finalizado el método Self-Training Obtención del número de datos etiquetados al iniciar el método Self-Training Obtención del número de datos no etiquetados al iniciar el método Self-Training Obtencion del número de iteraciones una vez finalizado el método Self-Training Entrenan el modelo Self-Training Devuelve un DataFrame con las predicciones utilizando el modelos previamente entrenado

Tabla D.4: Definición de la clase SelfTraining

la aplicación **sbt** [10] una vez instalado tendremos que proceder tal y como se indica en la Fígura D.1.

1. Creamos una carpeta donde exportaremos todos los ficheros desde Github.
2. Exportamos los ficheros directamente del Github de este proyecto utilizando su url [6]
3. Accedemos dentro del directorio donde tenemos todos los ficheros del proyecto (TFM-SemiSup)
4. Creamos el paquete .jar con el comando *sbt pacakge*

Clase	Descripción de clases	Extensión	Parámetros	Métodos	Descripción
Co-Training	Clase donde se ha implementado el algoritmo de SelfTraining la cual entrena con .fit el modelo y devuelve las predicciones de las clases con transform	ProbabilisticClassifier	ClasificadorBase ejemplo: NB, DT, LR...	setSemiSupervisedDataResults	Pasamos por parámetro la clase SemiSupervisedDataResults donde tendremos los resultados relacionados con los algoritmos semisupervisados
				setColumnName	Selección de la columna donde están las clases etiquetadas y no etiquetadas
				setThreshold	Selección del umbral de probabilidad donde se indicará si una instancia no etiquetada debe ser actualizada en el conjunto de datos etiquetados valores entre 0 y 1, se necesita seleccionar "Threshold," en setCriterion
				setCriterion	Selecciona si se quiere trabajar con threshold(umbral) o de lo contrario se decide un porcentaje de datos a etiquetar (kBest), este es un valor String, "Threshold," "kBest"
				setMaxIter	Número de iteraciones máximas según el algoritmo Self-Training, valores enteros
				setKbest	Porcentaje de valores No etiquetados que se desean etiquetar cuando se ha seleccionado "kBest," en SetCriterion
				getDataLabeledFinal	Obtención del número de datos etiqueta dos una vez finalizado el método Co-Training
				getUnDataLabeledFinal	Obtención del número de datos no etiqueta dos una vez finalizado el método Co-Training
				getDataLabeledIni	Obtención del número de datos etiqueta dos al iniciar el método Co-Training
				getUnDataLabeledIni	Obtención del número de datos no etiqueta dos al iniciar el método Co-Training
				getIter	Obtención del número de iteraciones una vez finalizado el método Co-Training
				fit/train	Entrenan el modelo Co-Training
				transform	Devuelve un DataFrame con las predicciones utilizando el modelos previamente entrenado

Tabla D.5: Definición de la clase Co-Training

5. dentro del directorio target/scala-X.XX tendremos finalmente el .jar

D.5. Como añadir el paquete .jar en DataBricks

En este apartado vamos a detallar paso por paso como añadir el .jar creado en [D.4](#).

1. Seleccionamos el icona Clusters (Figura [D.2](#)).

Clase	Descripción de clases	Extensión	Parámetros	funciones	Descripción
functionsSemiSupervised	Un object en el cual se ha definido todas las funciones utilizadas para las diferentes tareas de este trabajo. Ejemplo: validación cruzada, creación de pipelines, creación del dataFrame de resultados, cálculos de resultados...	No aplicable	No Aplicable	indexStringColumnsStagePipeline pipelineModelsSelf-Training pipelineModelsCo-Training generatorDataFrameResultadosSemiSuper crossValidation SupervisedAndSemiSupervisedResults	proceso de Featurization. Creacion de un pipeline stage con la conversión de las features (atributos) en valores numéricos y vectores para poder ser interpretado por Spark y el proceso de entrenamiento posterior Creación del pipeline final con todo los pipelines stages para posteriormente hacer un fit y transform con los datos de test Creación del pipeline final con todo los pipelines stages para posteriormente hacer un fit y transform con los datos de test Crea una plantilla de data frame donde se iran colocando todo los resultados posteriormente genera el calculo de resultados basándose en la validación cruzada Lanza los pipelines finales, calcula los resultados (accuracy, F1Score, number-LabeledIn...) y posteriormente los coloca en el dataFrame creado en 'generator-DataFrameResultadosSemiSuper'

Tabla D.6: Definición del objeto functionsSemiSupervised

```

root@iotPocVmLinux:~# mkdir /demoTfm 1
root@iotPocVmLinux:~# cd /demoTfm/
root@iotPocVmLinux:/demoTfm# git clone https://github.com/Dguipla/TFM-SemiSup
Cloning into 'TFM-SemiSup'...
remote: Enumerating objects: 314, done.
remote: Counting objects: 100% (314/314), done.
remote: Compressing objects: 100% (237/237), done.
remote: Total 462 (delta 91), reused 0 (delta 0), pack-reused 148
Receiving objects: 100% (462/462), 855.80 KiB | 17.83 MiB/s, done.
Resolving deltas: 100% (117/117), done.
root@iotPocVmLinux:/demoTfm# cd TFM-SemiSup/ 2
root@iotPocVmLinux:/demoTfm/TFM-SemiSup# ls
README.md build.sbt notebooks project src
root@iotPocVmLinux:/demoTfm/TFM-SemiSup# sbt package 3
[info] welcome to sbt 1.4.2 (adoptopenjdk Java 1.8.0_282)
[info] loading settings for project tfm-semisup-build from plugins.sbt ...
[info] loading project definition from /demoTfm/TFM-SemiSup/project
[warn] There may be incompatibilities among your library dependencies; run 'evicted' to see detailed eviction warnings.
[info] loading settings for project tfm-semisup from build.sbt ...
[info] set current project to spark_semisupervised_v4 (in build file:/demoTfm/TFM-SemiSup/)
[info] compiling 6 Scala sources to /demoTfm/TFM-SemiSup/target/scala-2.12/classes ...
[warn] there was one feature warning; re-run with -feature for details
[warn] one warning found
[success] Total time: 31 s, completed Mar 22, 2021 2:51:57 PM 4
root@iotPocVmLinux:/demoTfm/TFM-SemiSup# ls 5
README.md build.sbt notebooks project src target
root@iotPocVmLinux:/demoTfm/TFM-SemiSup# cd target/scala-2.12/
root@iotPocVmLinux:/demoTfm/TFM-SemiSup/target/scala-2.12# ls
classes spark_semisupervised_v4_2.12-1.0.0.jar update zinc 6
root@iotPocVmLinux:/demoTfm/TFM-SemiSup/target/scala-2.12#

```

Figura D.1: Creación del paquete .jar

2. Seleccionamos el icono del cluster donde queremos incluir la librería (Figura D.2).
3. Seleccionamos el icono Libraries (Figura D.3).
4. Seleccionamos Install New (Figura D.4).
5. Importamos nuestro .jar (Figura D.4).
6. Vemos que se ha completado la instalación una vez el ícono de Status esta en verde (Figura D.5).

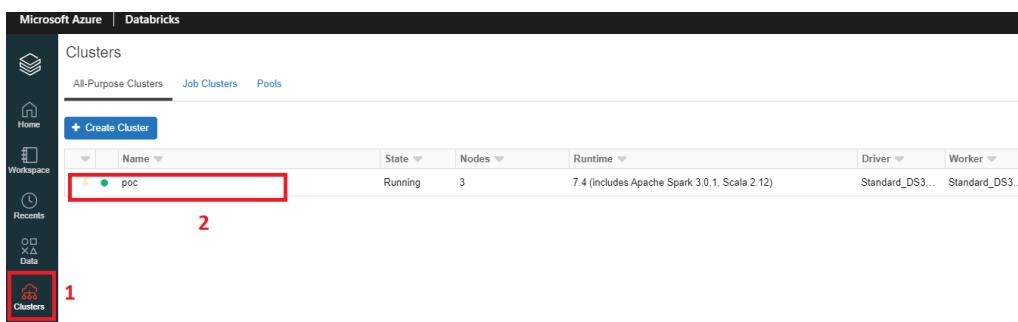


Figura D.2: Añadir paquete .jar paso-1

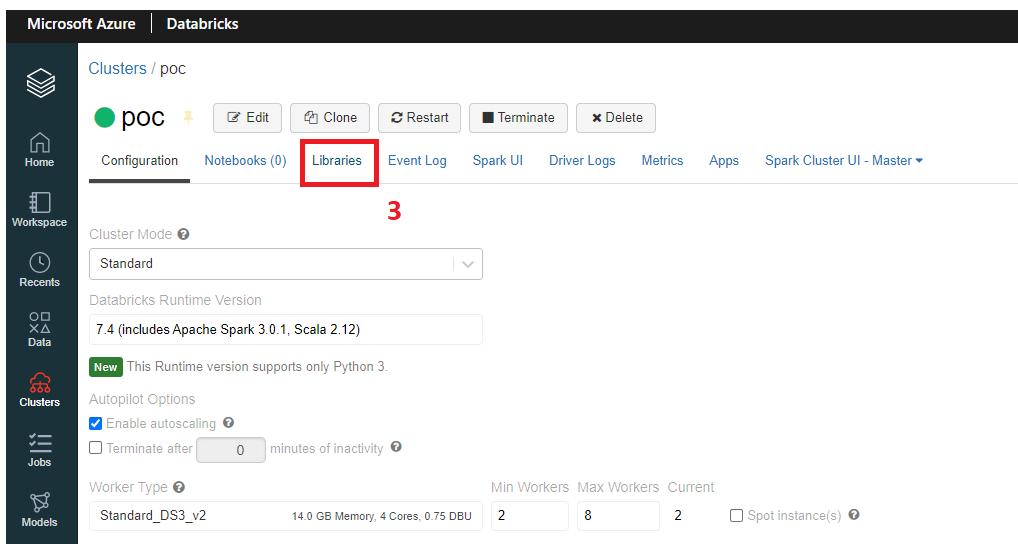


Figura D.3: Añadir paquete .jar paso-2

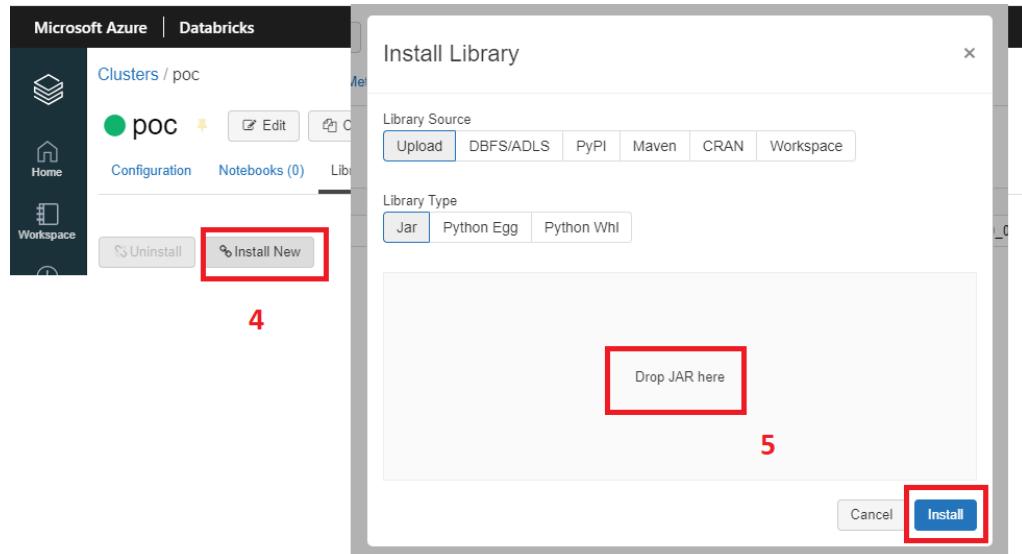


Figura D.4: Añadir paquete .jar paso-3

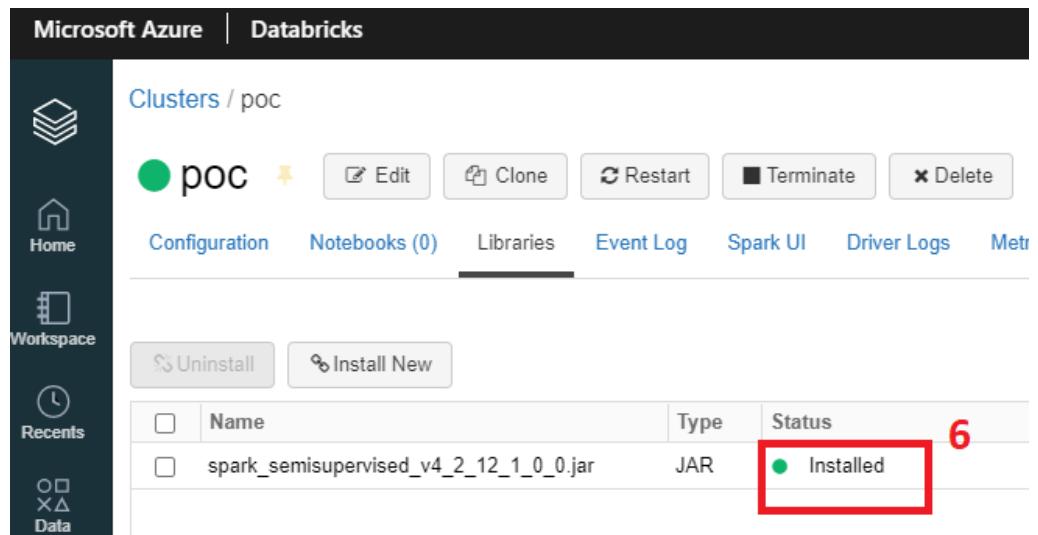


Figura D.5: Añadir paquete .jar paso-4

Apéndice E

Documentación de usuario

E.1. Introducción

En este apartado se va hacer una demo de como utilizar las librerías de semisupervisado (creadas en D), la idea principal es entender como funcionan para su posible utilización en trabajos futuros. También se va hacer una explicación (demo) de el dashboard con todos los resultados y como navegar sobre el.

E.2. Utilización de las librerías *org.apache.spark.ml.semisupervised._*

Como se ha comentado en la introducción, vamos ha realizar un ejemplo creando un modelo para los algoritmos semisupervisados y posteriormente vamos a predecir las clases para ver su funcionamiento. Para hacerlo más comprensible se va ir haciendo paso a paso, para ello se va a dividir en diferentes fases:

1. Importamos (Figura E.1) todas las librerías necesarias tanto para Self-Training y Co-Training como para hacer el proceso de desetiquetado (ya que para nuestros conjuntos de datos todas sus clases tienen etiquetas y tenemos que desetiquetarlas) o para obtener los resultados en relación a los datos etiquetados iniciales, finales... una vez finaliza el proceso.
También añadimos otras librerías necesarias para el proceso de *feature-
rization* como también para el clasificador base que para este ejemplo utilizaremos RF (Random Forest).

```

1 // import all the libraries from semisupervised
2 import org.apache.spark.ml.semisupervised.SelfTraining
3 import org.apache.spark.ml.semisupervised.CoTraining
4 import org.apache.spark.ml.semisupervised.Supervised
5 import org.apache.spark.ml.semisupervised.UnlabeledTransformer
6 import org.apache.spark.ml.semisupervised.FunctionsSemiSupervised._
7 import org.apache.spark.ml.semisupervised.SemiSupervisedDataResults
8
9 //import the base classifier (supervised) that we want to use
10 import org.apache.spark.ml.classification.RandomForestClassifier
11
12 //import other libraries for featurization process and pipelines
13 import org.apache.spark.sql.DataFrame
14 import org.apache.spark.ml.feature.StringIndexer
15 import org.apache.spark.ml.feature.StringIndexerModel
16 import org.apache.spark.ml.feature.VectorAssembler
17 import org.apache.spark.mllib.evaluation.MulticlassMetrics
18 import org.apache.spark.ml.{Pipeline, PipelineModel}
19 import org.apache.spark.ml.PipelineStage
20
21
22 import org.apache.spark.ml.semisupervised.SelfTraining
23 import org.apache.spark.ml.semisupervised.CoTraining
24 import org.apache.spark.ml.semisupervised.Supervised
25 import org.apache.spark.ml.semisupervised.UnlabeledTransformer
26 import org.apache.spark.ml.semisupervised.FunctionsSemiSupervised._
27 import org.apache.spark.ml.semisupervised.SemiSupervisedDataResults
28 import org.apache.spark.ml.classification.RandomForestClassifier
29 import org.apache.spark.sql.DataFrame
30 import org.apache.spark.ml.feature.StringIndexer
31 import org.apache.spark.ml.PipelineStage

```

Figura E.1: Importamos las librerías semisupervised.

2. Leemos el conjunto de datos con el que se va a trabajar, para este ejemplo utilizaremos `magic.csv`, posteriormente haremos el proceso de featurización el cual convertiremos las features (atributos) en un vector numérico y también convertiremos a valores numéricos las labels (clases), pare ello como se observa en la Figura E.2 se va a utilizar un pipeline con dos estados, primero convertimos las features y después para las labels como se indica en la linea 19. Posteriormente la ejecutaremos para hacer la transformación. Se ha optado en hacer la ejecución del pipeline por separado sin hacer todo el proceso en un único pipeline como se hace en este proyecto para que sea más simple el entendimiento y poder explicar todos los pasos de lo contrario seria mucho más abstracto, no obstante es más eficiente hacer un diseño embebido en un solo pipeline como se ha hecho en este trabajo (apartado C).
3. Hacemos una división en datos de entrenamiento y test en un 75 % y un 25 % (Figura E.3).
4. Como se ha comentado en la introducción se tiene que desetiquetar parte de las clases para ello utilizamos la clase `UnlabeledTransformer()`

```

1  /** read data*/
2  val data = "magic.csv"
3  var PATH="dbfs:/FileStore/tables/"
4  var dataDF = spark.read.format("csv")
5    .option("sep", ",")
6    .option("inferSchema", "true")
7    .option("header", "true")
8    .load(PATH + data)
9  dataDF = dataDF.na.drop()
10
11 /**featurization*/
12 val dataFeatures = dataDF.columns.diff(Array(dataDF.columns.last))
13 val dataFeaturesLabelPipeline = new VectorAssembler().setOutputCol("features").setInputCols(dataFeatures)
14
15 /**StringIndexer for labels to change from categorical to numerical (double)*/
16 val indexClassPipeline = new StringIndexer().setInputCol(dataDF.columns.last).setOutputCol("label").setHandleInvalid("skip")
17
18 /**pipelineFeaturization*/
19 val featurizationPipeline= new Pipeline().setStages(Array(dataFeaturesLabelPipeline, indexClassPipeline))
20
21 /**convert the pipeline for featurization --> new structure features (double of vector) and labels (doubles)*/
22 val dataFeat = featurizationPipeline.fit(dataDF).transform(dataDF).select("features", "label")
23

```

▶ (4) Spark Jobs

- ▶ dataDF: org.apache.spark.sql.DataFrame = [FLength: double, FWidth: double ... 9 more fields]
- ▶ dataFeat: org.apache.spark.sql.DataFrame = [features: udt, label: double]

data: String = magic.csv
PATH: String = dbfs:/FileStore/tables/
dataDF: org.apache.spark.sql.DataFrame = [FLength: double, FWidth: double ... 9 more fields]
dataDF: org.apache.spark.sql.DataFrame = [FLength: double, FWidth: double ... 9 more fields]

Figura E.2: Leemos del conjunto de datos y realizamos el proceso de featurization.

```

1  /** split data for training and test 75% and 25%*/
2  val dataSplited = dataFeat.randomSplit(Array(0.75, 1 - 0.25), seed = 11L)
3  val dataTraining = dataSplited(0)
4  val dataTest = dataSplited(1)
5

```

▶ dataTraining: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: udt, label: double]
▶ dataTest: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: udt, label: double]

dataSplited: Array[org.apache.spark.sql.Dataset[org.apache.spark.sql.Row]] = Array([features: vector, label: double], [features: vector, label: double])
dataTraining: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: vector, label: double]
dataTest: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [features: vector, label: double]

Figura E.3: División en datos de entrenamiento y test.

como se puede ver en la Figura E.4 donde también seleccionamos el 10 % de datos etiquetados.

```

1  /** remove the labels in this case 85% only 15% of data with label*/
2  val dataLabeledUnLabeledTraining = new UnlabeledTransformer().setPercentage(0.15).setColumnName("label").transform(dataTraining)

```

▶ dataLabeledUnLabeledTraining: org.apache.spark.sql.DataFrame = [features: udt, label: double]
dataLabeledUnLabeledTraining: org.apache.spark.sql.DataFrame = [features: vector, label: double]

Figura E.4: Desetiquetamos las clases.

5. Finalmente (Figura E.5) creamos la instancia de la clase Self-Training con RF como clasificador base, posteriormente generamos el modelo entrenando con los datos de entrenamiento (training), para finalizar hacemos un **transform** para obtener las predicciones y ver el acierto/accuracy. Para este caso es un 0,799 (79,9 %).

```

1  /** selfTraining model */
2  // results regarding the initial labeled data, final labeled, porcentage labeled ...
3  val resultsSemi = new SemiSupervisedDataResults()
4
5  //model training with a threshold of 0.9
6  val modelSelfTraining = new SelfTraining(new RandomForestClassifier())
7  .setSemiSupervisedDataResults(resultsSemi)
8  .setThreshold(0.9)
9  .fit(dataLabeledUnLabeledTraining)
10
11 /** predictions with dataTest*/
12 val predictions = modelSelfTraining.transform(dataTest).select("prediction", "label")
13
14 /** accuracy metrics */
15 val predictionsAndLabelsRDD = predictions.rdd.map(row => (row.getDouble(0), row.getDouble(1)))
16 val metrics= new MulticlassMetrics(predictionsAndLabelsRDD)
17 val accuracy = metrics.accuracy
18 val error = 1 - accuracy
19
20
21 (65) Spark Jobs
22 predictions: org.apache.spark.sql.DataFrame = [prediction: double, label: double]
23 resultsSemi: org.apache.spark.ml.semisupervised.SemiSupervisedDataResults = org.apache.spark.ml.semisupervised.SemiSupervisedData
24 modelSelfTraining: org.apache.spark.ml.classification.RandomForestClassificationModel = RandomForestClassificationModel: uid=rfc_
25 predictions: org.apache.spark.sql.DataFrame = [prediction: double, label: double]
26 predictionsAndLabelsRDD: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[6746] at map at command-3060274697655634:1
27 metrics: org.apache.spark.mllib.evaluation.MulticlassMetrics = org.apache.spark.mllib.evaluation.MulticlassMetrics@6dfbb86c
28 accuracy: Double = 0.7996600807308264
29 error: Double = 0.2003399192691736

```

Figura E.5: Entrenamos el modelo para Self-Training con RF y vemos los resultados obtenidos con este modelo.

6. Para acabar también mostramos los resultados en relación a los datos etiquetados y no etiquetados al inicio y final del proceso para entender como funciona el modelo y si estamos sacando ventaja, para ello se utiliza la clase **SemiSupervisedDataResults**. Para este caso, podemos observar estos resultados en la Figura E.6.

```

1  /** check the data labeled at the end of the process*/
2  println ("data labeled final " + resultsSemi.dataLabeledFinal)
3  println ("data labeled initial " + resultsSemi.dataLabeledIni)
4  println ("data unlabeled final " + resultsSemi.dataUnDatalabeledFinal)
5  println ("data unlabeled initial " + resultsSemi.dataUnLabeledIni)

```

```

data labeled final 6368
data labeled initial 1455
data unlabeled final 3238
data unlabeled initial 8151

```

Figura E.6: Obtenemos los resultados de los datos etiquetado y no etiquetados al inicio y final del proceso de Self-Training.

7. Para el caso de Co-Training sería todo igual simplemente instanciaríamos la clase de Co-Training por Self-Training (Figura E.5), observar Figura E.7 línea 9. Para este caso se ve que el resultado en relación al acierto/accuracy es ligeramente superior, para este caso es 0,808 (80,08 %).

```
1  /** CoTraining model */
2 // results regarding the initial labeled data, final labeled, percentage labeled ...
3 val resultsSemi = new SemiSupervisedDataResults ()
4
5 //model training with a threshold of 0.9
6 val modelSelfTraining = new CoTraining(new RandomForestClassifier())
7   .setSemiSupervisedDataResults(resultsSemi)
8   .setThreshold(0.9)
9   .fit(dataLabeledUnLabeledTraining)
10
11 /** predictions with dataTest*/
12 val predictions = modelSelfTraining.transform(dataTest).select("prediction", "label")
13
14 /** accuracy metrics */
15 val predictionsAndLabelsRDD = predictions.rdd.map(row => (row.getDouble(0), row.getDouble(1)))
16 val metrics= new MulticlassMetrics(predictionsAndLabelsRDD)
17 val accuracy = metrics.accuracy
18 val error = 1 - accuracy

▶ (39) Spark Jobs
▶ predictions: org.apache.spark.sql.DataFrame = [prediction: double, label: double]
resultsSemi: org.apache.spark.ml.semisupervised.SemiSupervisedDataResults = org.apache.spark.ml.semisupervised.SemiSupervisedDataResults@33333333
modelSelfTraining: org.apache.spark.ml.classification.RandomForestClassificationModel = RandomForestClassificationModel@33333333
predictions: org.apache.spark.sql.DataFrame = [prediction: double, label: double]
predictionsAndLabelsRDD: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[1530] at map at command-3060274697
metrics: org.apache.spark.mllib.evaluation.MulticlassMetrics = org.apache.spark.mllib.evaluation.MulticlassMetrics@49afff51
accuracy: Double = 0.8086891863182494
error: Double = 0.19131081368175062
```

Figura E.7: Entrenamos el modelo para Co-Training con RF y vemos los resultados obtenidos con este modelo.

Bibliografía

- [1] Arquitectura spark. <https://data-flair.training/blogs/how-apache-spark-works/>.
- [2] Clase en spark pipelinestage. <https://spark.apache.org/docs/latest/api/java/index.html?org/apache/spark/ml/PipelineStage.html>.
- [3] Dataframes información. <https://spark.apache.org/docs/latest/sql-programming-guide.html>.
- [4] Decision tree algorithm in spark. <https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-tree-classifier>.
- [5] Explicación de las métricas pr y auc. <https://www.youtube.com/watch?v=fF0MvCPsEp8>.
- [6] Github del tfm. <https://github.com/Dguipla/TFM-SemiSup>.
- [7] herramienta y datos keel. <https://sci2s.ugr.es/keel/description.php>.
- [8] Implementaciones de métodos ssc en github. https://github.com/hbq1/Semi-supervised_Learning_in_Apache_Spark.
- [9] Implementaciones de métodos ssc en github 2. https://github.com/hbq1/Semi-supervised_Learning_in_Apache_Spark.
- [10] Información sobre sbt y proceso de instalación. <https://www.scala-sbt.org/1.x/docs/Setup.html>.

- [11] Logistic regression algorithm in spark. <https://spark.apache.org/docs/latest/ml-classification-regression.html#logistic-regression>.
- [12] Métricas mllib. <https://spark.apache.org/docs/1.6.1/mllib-evaluation-metrics.html>.
- [13] Naive bayes algorithm in spark. <https://spark.apache.org/docs/latest/ml-classification-regression.html#naive-bayes>.
- [14] Overleaf como editor latex para la memoria. <https://www.overleaf.com/>.
- [15] Random forest algorithm in spark. <https://spark.apache.org/docs/latest/sql-programming-guide.html>.
- [16] Rdd data. <https://spark.apache.org/docs/latest/ml-guide.html>.
- [17] Semi-supervised learning... the great unknown. <https://business.blogthinkbig.com/semi-supervised-learning-the-great-unknown/>.
- [18] Zenhub para la gestión del proyecto. <https://www.zenhub.com/>.
- [19] Kristin P Bennett, Ayhan Demiriz, and Richard Maclin. Exploiting unlabeled data in ensemble methods. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–296, 2002.
- [20] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998.
- [21] Nikos Fazakis, Stamatis Karlos, Sotiris Kotsiantis, and Kyriakos Sgarbas. Self-trained lmt for semisupervised learning. *Computational intelligence and neuroscience*, 2016, 2016.
- [22] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *ICML*, pages 327–334. Citeseer, 2000.
- [23] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *ICML*, pages 327–334. Citeseer, 2000.

- [24] Tao Guo and Guiyang Li. Improved tri-training with unlabeled data. In *Software engineering and knowledge engineering: Theory and practice*, pages 139–147. Springer, 2012.
- [25] Mohamed Farouk Abdel Hady and Friedhelm Schwenker. Semi-supervised learning. In *Handbook on Neural Information Processing*, pages 215–239. Springer, 2013.
- [26] Vasileios Iosifidis and Eirini Ntoutsi. Large scale sentiment learning with limited labels. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1823–1832, 2017.
- [27] Zhen Jiang, Shiyong Zhang, and Jianping Zeng. A hybrid generative/discriminative method for semi-supervised classification. *Knowledge-Based Systems*, 37:137–145, 2013.
- [28] Zhen Jiang, Shiyong Zhang, and Jianping Zeng. A hybrid generative/discriminative method for semi-supervised classification. *Knowledge-Based Systems*, 37:137–145, 2013.
- [29] Ming Li and Zhi-Hua Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(6):1088–1098, 2007.
- [30] Yu-Feng Li and Zhi-Hua Zhou. Improving semi-supervised support vector machines through unlabeled instances selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 2011.
- [31] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D Cubuk, and Ian J Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. *arXiv preprint arXiv:1804.09170*, 2018.
- [32] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information systems*, 42(2):245–284, 2015.
- [33] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.
- [34] Joaquin Vanschoren. Meta-learning: A survey, 2018.

- [35] Jiao Wang, Si-wei Luo, and Xian-hua Zeng. A random subspace method for co-training. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 195–200. IEEE, 2008.
- [36] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.
- [37] Yan Zhou and Sally Goldman. Democratic co-learning. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 594–602. IEEE, 2004.
- [38] Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*, 17(11):1529–1541, 2005.
- [39] Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*, 17(11):1529–1541, 2005.
- [40] Zhi-Hua Zhou and Ming Li. Pseudo-labeling to deal with small datasets what, why and how?, 2019.