UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Estudio de métodos de selección de instancias en aprendizaje Semi-Supervisado y aplicación web de MLaaS

IS-SSL DNX

Presentado por Daniel Puente Ramírez
en Universidad de Burgos — 24 de mayo de 2022
Tutor: Dr. Álvar Arnaiz González

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Álvar Arnaiz González, profesor del Departamento de Ingeniería Informática, Área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Daniel Puente Ramírez, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Estudio de métodos de selección de instancias en aprendizaje Semi-Supervisado y aplicación web de MLaaS».

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, 24 de mayo de 2022

Vº. Bº. del Tutor:

D. Álvar Arnaiz González

Resumen

Aquellos algoritmos más comúnmente utilizados en la literatura no acostumbran a encontrarse disponibles a través de bibliotecas existentes que los recojan, induciendo la necesidad de que el científico de datos que los necesite, deba implementarlos y validarlos antes de poder hacer uso de los mismos.

El primer objetivo del proyecto es la creación de dos bibliotecas, las cuales recojan los algoritmos más comunes de selección de instancias, y de aprendizaje semi-supervisado. Las bibliotecas se validarán y se comprobará su integridad, además se realizará una experimentación completa con el fin de validar la hipótesis:
«¿Se obtiene una selección más segura en aprendizaje semi-supervisado gracias a la aplicación de métodos de selección de instancias?».
Las bibliotecas quedarán a disposición de la comunidad *Open source* tras la finalización del proyecto.

El segundo objetivo es la integración de sendas bibliotecas en `UBUMLaaS`, aplicación de aprendizaje máquina en la nube, perteneciente al grupo de investigación ADMIRABLE de la Universidad de Burgos. Además de su modernización y creación del nuevo rol de administrador del sistema, con todas las opciones y vistas que con ello conlleva; finalmente se añadirá una vista de monitorización del sistema en tiempo real y visualización de estadísticas.

Descriptores

selección de instancias, aprendizaje semi-supervisado, aprendizaje semi-supervisado seguro, aprendizaje máquina como servicio

Abstract

Those algorithms most commonly used in the literature are not usually available through existing libraries that collect them, inducing the need for the data scientist who needs them to implement and validate them before being able to make use of them.

The first objective of the project is the creation of two libraries, which collect the most common algorithms for instance selection and semi-supervised learning. The libraries will be validated and checked for completeness, and a complete experimentation will be carried out in order to validate the hypothesis:
Is safer selection obtained in semi-supervised learning as a result of the application of instance selection methods?.
The libraries will be made available to the *Open source* community after the completion of the project.

The second objective is the integration of both libraries in `UBUMLaaS`, a machine learning application in the cloud, belonging to the ADMIRABLE research group of the University of Burgos. In addition to its modernization and creation of the new role of system administrator, with all the options and views that this entails; finally a real-time system monitoring view and statistics visualization will be added.

Keywords

instance selection, semi-supervised learning, safe semi-supervised learning, machine learning as a service

Índice general

Índice de figuras

Índice de tablas

Introducción

Actualmente no se dispone de ninguna biblioteca en Python que facilite a los científicos de datos aplicar técnicas de Selección de Instancias sobre grandes conjuntos de datos, siendo esta una carencia detectada y teniendo en mente el auge que posee el lenguaje de programación, se propone la creación de un biblioteca que recoja aquellos algoritmos más comúnmente utilizados en la literatura.

Lo mismo sucede con los algoritmos de aprendizaje semi-supervisado, la no disponibilidad de estos en un momento en el que es un campo que está siendo investigado por gran parte de la comunidad científica enfocada en *Machine Learning*, retrasa y dificulta la investigación y la reproductibilidad de experimentos.

Ambas bibliotecas propuestas en este trabajo se encuentran a disposición de quien las necesite para su trabajo, así pues su licencia es BSD 3-Clause. La intención perseguida no es solo el crear un proyecto y que este sea descontinuado una vez se finalice el proyecto, sino que cualquiera pueda seguir expandiendo las bibliotecas con nuevos algoritmos de forma que sea un proyecto capaz de crecer y ser mantenido. De tal manera que conformen la primera aportación formal del desarrollador a la comunidad *Open source*.

Se utilizarán ambas bibliotecas con el fin de realizar una experimentación en el campo del aprendizaje semi-supervisado seguro, pretendiendo validar la hipótesis de si se obtiene una mejor selección gracias a la aplicación de métodos de selección de instancias en el proceso del aprendizaje semi-supervisado.

Por otro lado reside el *Machine Learning as a Service*, MLaaS. El desarrollo de un producto para convertirlo en un servicio completo en la nube ha visto el aumento de nuevos servicios, entre los que se encuentran el *PaaS*, *IaaS*, *SaaS*, y más recientemente, *MLaaS*. Con una tendencia creciente de trasladar el almacenamiento de datos a la nube, mantenerlos y obtener los mejores conocimientos de ellos, *MLaaS* surge como un gran aliado gracias a su capacidad de proporcionar estas soluciones a un coste reducido [9].

La Universidad de Burgos, más concretamente el grupo de investigación ADMIRABLE, posee su propia aplicación de *MLaaS*, bajo el nombre de `UBUMLaaS`. Es objetivo de este proyecto su modernización, adaptación para dar una primera cabida a algoritmos de aprendizaje semi-supervisado, así como su ampliación de forma que como cualquier plataforma, disponga de capacidades propias de administración y visualización de estadísticas.

El estado inicial de `UBUMLaaS` requiere de constantes accesos a la base de datos para realizar modificaciones sobre usuarios y sus parámetros, es por ello que se quiere realizar una «parte» de administración para que usuarios con un nuevo rol de administrador puedan realizar las operaciones pertinentes de forma correcta.

Además, se integran opciones de visualización estadística tanto para usuarios como para administradores, siendo reportadas estadísticas de uso personales o del sistema, respectivamente. Para aquellos usuarios con la suerte de ser administradores, se les proporciona una vista del estado en tiempo real del sistema, con el fin de poder realizar un seguimiento y toma de decisiones acorde a lo que se pueda visualizar.

1.1. Estructura de la memoria

La memoria posee la siguiente estructura:

- **Introducción.** Descripción del proyecto y estructura de la documentación.
- **Objetivos del proyecto.** Explicación de los objetivos principales que sigue el proyecto.
- **Conceptos teóricos.** Explicación de aquellos conceptos cuya comprensión es clave para poder comprender el proyecto desarrollado.

- **Técnicas y herramientas.** Breve explicación de cada técnica, metodología, y herramienta utilizada para el desarrollo del proyecto.
- **Aspectos relevantes.** Exposición de aquellos aspectos destacables y que tuvieron lugar a lo largo de la realización del proyecto. Además se incluyen los resultados de la investigación realizada.
- **Trabajos relacionados.** Estado del arte de aquellos trabajos y proyectos relacionados con la selección de instancias, el aprendizaje semi-supervisado, y los *MLaaS*.
- **Conclusiones y Líneas de trabajo futuras.** Conclusiones alcanzadas tras la realización del proyecto, y siguientes pasos a dar tanto en investigación como en mejora de los diferentes productos desarrollados.

El documento de anexos posee la siguiente estructura:

- **Plan del proyecto software.** Exposición de la planificación temporal y los estudias de viabilidad económica y legal.
- **Especificación de requisitos del software.** Exposición en detalle de los objetivos del proyecto, así como el catálogo de requisitos y la especificación de requisitos funcionales y no funcionales.
- **Especificación de diseño.** Explicación de las decisiones seguidas para cumplir con los objetivos del proyecto. Y las principales características del diseño.
- **Documentación técnica de programación.** Exposición de toda aquella información relevante para futuros desarrolladores encargados de continuar con alguno de los proyectos.
- **Documentación de usuario.** Guía la cual puede seguir cualquier usuario para poder hacer uso del proyecto.

1.2. Materiales adjuntos

Los materiales que se adjuntan con la memoria son:

- Biblioteca de algoritmos de selección de instancias y aprendizaje semi-supervisado, `IS-SSL`.

- Aplicación `UBUMLaaS` para su despliegue directo.
- Contenedor `Docker` con `UBUMLaaS` ya desplegado.
- Resultados de la experimentación.

Los siguientes recursos son accesibles a través de Internet:

- Repositorio del proyecto `IS-SSL` [52].
- Biblioteca de algoritmos de selección de instancias en `PyPI` [51].
- Biblioteca de algoritmos de aprendizaje semi-supervisado en `PyPI` [53].
- Repositorio del proyecto `UBUMLaaS` [54].
- Contenedor `Docker` con `UBUMLaaS` desplegado [50].

Objetivos del proyecto

Los principales objetivos del proyecto son cuatro:

1. Diseño e implementación de una biblioteca con los algoritmos de selección de instancias más comunes en la literatura.

2. Diseño e implementación de una biblioteca con una serie de algoritmos de aprendizaje semi-supervisado.

3. Realización de una experimentación en el campo de investigación del aprendizaje semi-supervisado seguro. Descubrir el efecto de la aplicación de diferentes métodos de selección de instancias.

4. Integración de las bibliotecas con la plataforma de `MLaaS` de la Universidad de Burgos (`UBUMLaaS`).

5. Diseño y puesta en producción de la parte de administración de `UBUMLaaS`.

El enfoque que se le debe dar a las bibliotecas, en adelante `IS-SSL`[1], tanto de selección de instancias como de aprendizaje semi-supervisado, deberá permitir de manera sencilla la inclusión o añadido de nuevos algoritmos en un futuro, no siendo necesaria realizar grandes refactorizaciones para ello. Mediante ello se obtendrá un producto escalable y con un mantenimiento relativamente sencillo.

[1] *Instance Selection - Semi-Supervised Learning.*

`UBULMLaaS` fue un proyecto desarrollado por el grupo de investigación ADMIRABLE y se paralizó en 2019, por lo que necesitará una actualización de bibliotecas, interfaz gráfica, seguridad y actualización de la base de datos; entre otras cosas. Independientemente de los cambios, debe primar la sencillez de uso que la aplicación, de forma que la curva de aprendizaje sea mínima.

Objetivos técnicos

Además de lo anteriormente mencionado, el proyecto cuenta con una serie de objetivos técnicos que se pueden resumir en:

- Los algoritmos imeplementados en `IS-SSL` deberán seguir la guía de estilo de *Scikit-Learn* [62], permitiendo a la comunidad científica acostumbrada al uso de la mencionada biblioteca en `Python`, hacer uso de `IS-SSL` de igual manera.
- Los algoritmos deberán de ser validados de alguna manera, ya sea con la literatura o mediante pares, para asegurar un correcto funcionamiento.
- `UBUMLaaS` deberá tener distintos tipos o categorías de usuarios, debiendo dejar «la puerta abierta» a nuevos tipos de usuarios en el futuro.
- `UBUMLaaS` podrá ser portado y desplegado sobre *bare metal* o mediante contenedores de Docker en cualquier sistema compatible.
- `UBUMLaaS` debe mantener todas sus funcionalidades previas a este proyecto.
- `UBUMLaaS` mostrará estadísticas generadas en tiempo real, se deberá de sortear la problemática de la concurrencia de acceso a registros de la base de datos, así como ficheros temporales.
- `UBUMLaaS` posee su propia API REST escrita en Python y emplea el *framework web* Flask. No se deberá sobrecargar su uso, la carga de trabajo deberá estar balanceada entre cliente y servidor.

Conceptos teóricos

El proyecto tiene una relación directa con la minería de datos y los conceptos que lo rodean.

3.1. Aprendizaje automático (*machine learning*)

En [60] se define el aprendizaje automático (*machine learning*) como una rama dentro del campo de la Inteligencia Artificial que proporciona a los sistemas la capacidad de aprender y mejorar de manera automática, a partir de la experiencia. Estos sistemas transforman los datos en información, y con esta información pueden tomar decisiones. Este tipo de modelos se crean a base del uso masivo de datos. Cuando se dispone de los datos suficientes para entrenar un modelo comienza el proceso de aprendizaje. El objetivo de este aprendizaje es descubrir patrones ocultos en los datos. En muchas ocasiones el resultado del aprendizaje, el modelo, es una función que dadas unos datos de entrada clasifica o predice correctamente una salida. Como se puede ver en la Figura 3.1 el aprendizaje automático, *machine learning*, posee diferentes aproximaciones, siendo la interfaz diferenciadora entre ellas la forma de uso de las instancias.

Aprendizaje supervisado

El aprendizaje automático puede ser resumido como «aprender de ejemplos». Al programa se le proporcionan dos conjuntos de datos, uno de entrenamiento y otro de validación [36]. El objetivo es simple, debe de «aprender» en función del conjunto de datos etiquetado proporcionado

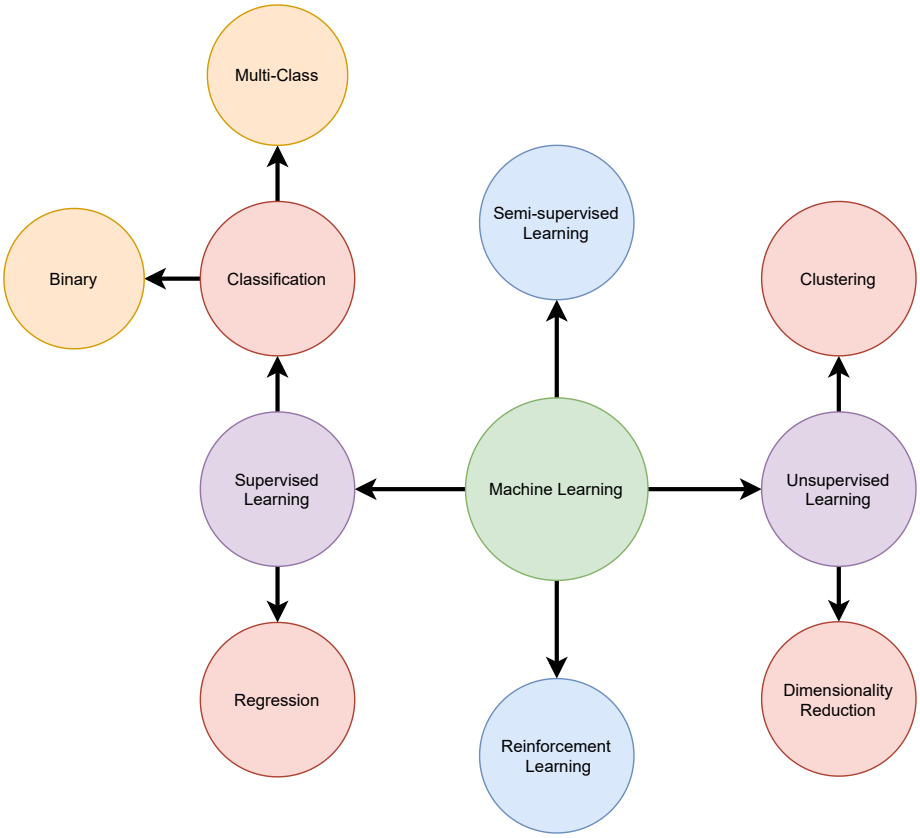


Figura 3.1: *Machine learning overview* [66].

como entrenamiento para posteriormente identificar las correspondientes etiqueta/s de cada instancia del conjunto de validación con la mayor precisión posible.

Dependiendo del tipo de etiqueta, en el aprendizaje supervisado hay dos modelos [33].

1. **Modelos de clasificación.** Producen como salida una etiqueta discreta, *i.e.* una etiqueta dentro de un conjunto finito de etiquetas, habitualmente suelen ser o binarias $[0, 1]$, $[sí, no]$... o multi-etiqueta, donde por ejemplo los valores pueden variar $[0...n]$, *i.e.* no tienen que ser estrictamente numéricas, pudiendo ser categóricas como por ejemplo $\{coche, moto, barco\}$. En los modelos de clasificación multi-etiqueta es habitual que el clasificador trabaje con selección de varias etiquetas para la misma muestra, no estando restringido a una única [70].

   Entre los algoritmos de clasificación más frecuentes encontramos:

- Regresión logística.
- *Support Vector Machine, SVM.*
- Redes neuronales.
- Clasificador Naïve Bayes.
- Árbol de decisión.
- Análisis discriminante.
- *K* vecinos más cercanos, *KNN.*
- Clasificación con ensembles.

2. **Modelos de regresión.** Producen como salida un valor real, numérico. Suelen ser soluciones continuas. De igual manera si se quieren obtener varios resultados de una muestra, se denomina multi-*output* [11].

   Entre los algoritmos de regresión más frecuentes encontramos:

   - Regresión lineal.
   - Regresión no lineal.
   - Modelo lineal generalizado.
   - Árbol de decisión.
   - Redes neuronales.
   - Regresión con procesos gaussianos.
   - Regresión con *support vector machines.*
   - Regresión con ensembles.

Aprendizaje no supervisado

En la Sección 3.1 se comenta que, los modelos para que «aprendan» los patrones que se encuentran en los conjuntos de datos, necesitan tener un conjunto de datos etiquetado correctamente para extraer la información de ese conjunto. Pero en los problemas del mundo real no siempre se tienen infinidad de datos disponibles etiquetados correctamente, o simplemente es un proceso muy laborioso y costoso económicamente.

Para solventar este problema se cuenta con el aprendizaje no supervisado [8], mediante esta técnica no es necesario proporcionar al modelo datos etiquetados. Por definición, el algoritmo encargado de entrenar el modelo «aprenderá» los datos sin conocimiento previo. Para ello el modelo

se basará en los datos que tiene disponibles y en la codificación del algoritmo para descubrir los patrones que se encuentren en los datos.

Debido a la forma de trabajar del aprendizaje no supervisado, desde el primer momento en el que el algoritmo tiene los datos comienza a reportar salidas, describiendo la información y categorizando lo que encuentra en los mismos.

Principalmente existen dos técnicas de aprendizaje no supervisado.

1. ***Clustering*** [43]. Proceso por el cual se dividen los datos no clasificados en grupos aparentemente similares. Cuando se identifican datos con algún parecido entre sí, son agrupados. Permite clasificar e identificar atributos únicos de los datos con los que clasificarlos.

   Un proceso habitual de agrupamiento es el uso de *K-means*, $K \in \mathbb{R}$, donde se indica en K cuántos *clusters* o grupos se han de identificar en los datos.

   Con los datos agrupados el proceso de análisis de éstos puede comenzar. En ocasiones si el número de grupos detectados es muy alto, se pueden encontrar grupos o *clusters* irrelevantes, permitiendo a los científicos de datos eliminar esos datos que los forman, reduciendo la dimensionalidad [12].

2. **Reducción de la dimensionalidad** [38]. El aprendizaje se basa en instancias/ejemplos y éstos a su vez están formados por atributos o características, lo cual permite su clasificación, valga la redundancia. Cuando los conjuntos de datos poseen múltiples características, más difícil resulta su clasificación. Es por ello que resulta útil identificar aquellos atributos que están fuertemente interrelacinados entre sí para eliminar todos menos un atributo, reduciendo la dimensionalidad [38].

Aprendizaje semi-supervisado

Según [82] *Semi-Supervised Learning* se define como una forma de entrenamiento de modelos el cual usa tanto datos etiquetados como no etiquetados, *i.e.* si no sería un aprendizaje supervisado, o no supervisado.

El uso de aprendizaje semi-supervisado se caracteriza por ser menos costoso que el supervisado, ya que este último necesita que todo el conjunto de datos que va a utilizar para aprender esté etiquetado, y ese proceso es normalmente costoso (en tiempo y en recursos). Luego, obtiene mejores

resultados en menor tiempo que el aprendizaje no supervisado. Conseguir datos sin etiquetar es una tarea muy sencilla, mientras que conseguir conjuntos de datos etiquetados es un proceso complejo y actualmente no hay «de todo».

Para que el aprendizaje sea fructuoso requiere que las instancias se encuentren inter-relacionadas entre sí por alguna de sus características [34] indica las siguientes suposiciones que se dan en el aprendizaje semi-supervisado.

1. **Continuity** o continuidad. Se asume que los objetos cercanos entre sí se encontrarán en el mismo *cluster* o grupo de etiquetas.
2. ***Clustering*** o agrupamiento. Las instancias son divididas en diferentes grupos discretos, compartiendo todos los elementos de un *cluster* la misma etiqueta.
3. ***Manifold*** o variedad. Se emplea el uso de distancias y funciones de densidad de forma que las instancias se encuentran en colectores con menos dimensiones que el espacio de entrada.

Dentro de las *best practices* en *semi-supervised learning* se encuentran el uso de diferentes modelos de redes neuronales para el entrenamiento [67].

3.2. Algoritmos de aprendizaje semi-sepervisado

A continuación se van a presentar los algoritmos implementados en este trabajo dentro del aprendizaje semi-supervisado.

Self-training

Los métodos de auto-etiquetado (*self-training*), son uno de los métodos más sencillos de pseudo-etiquetado que existen [68]. Consisten en un único clasificador el cual utiliza aprendizaje supervisado, el clasificador es entrenado con datos etiquetados conocidos al comienzo del entrenamiento y por los que se van conociendo a medida que pasan las iteraciones. Es decir, el clasificador inicialmente es entrenado con aquellos datos para los que se posea una etiqueta, y en sucesivas iteraciones se le reentrenará con

el nuevo conjunto de datos etiquetado, que es el resultante de añadir a los datos etiquetados previos, aquellas predicciones con un elevado *condifence level* [23].

Yarowsky [80] en 1995 propuso la primera versión de *Self-training*, desde entonces numerosas aproximaciones han sido realizadas, modificando la utilización del conjunto de datos etiquetado, los nuevos datos, etcétera. El diseño que se le puede dar y los campos de aplicación del mismo son muy variados.

El procedimiento de selección de qué datos son pseudo-etiquetados es de vital importancia, puesto que determinará qué datos acaban en el conjunto de datos etiquetado de sucesivas iteraciones. Siendo este un proceso iterativo y no incremental, ya que la probabilidad de etiquetado de los datos es re-calculada en cada iteración. De no serlo sería una apoximación a *expectation-maximization* [21].

Co-Training

Blum [10] en 1998 propuso el *Co-Training* para conjuntos de datos compuestos por datos separables en dos vistas. Bajo la presunción de que con unos pocos datos etiquetados y diferentes clases que aportan información, se pueden entrenar dos algoritmos de aprendizaje por separado para posteriormente añadir al conjunto de datos etiquetados aquellas predicciones con un mayor intervalo de confianza (*confidence level*).

Las dos características del problema mencionadas anteriormente, disponibilidad de datos etiquetados y no etiquetados, y la disponibilidad de dos «tipos» diferentes de conocimiento sobre los ejemplo; aproximan a la siguiente estrategia de aprendizaje. Se desea encontrar los predictores débiles basados en cada tipo de información utilizando un pequeño conjunto inicial de instancias etiquetadas, seguidamente, utilizando los datos no etiquetados se intenta hacer un *bootstrap* a partir de esos «malos» predictores. Este tipo de *bootstrapping* es el denominado *Co-Training*, y posee una estrecha relación con el *bootstrapping* a partir de datos incompletos en el marco de la maximización de expectativas [28, 55].

Algoritmo 1: *Co-Training*

Entrada: Conjunto de entrenamiento $L\{(x_i, y_i)\}_{i=1}^{l}$ y $U\{x_j\}_{j=l+1}^{l+u}$ de datos etiquetados y no etiquetados, respectivamente

Entrada: p, n, k, u, datos positivos a seleccionar, los negativos, iteraciones, tamaño *pool* inicial

1 $U' \leftarrow U[u]$ `/* u instancias aleatorias de U */`
2 **for** k **do**
3 Usar L para entrenar un clasificador h_1 utilizando x
4 Usar L para entrenar un clasificador h_2 utilizando y
5 h_1 clasifica U'
6 h_2 clasifica U'
7 Seleccionar las p y n instancias con mayor intervalo de confianza de cada clasificador
8 Añadirlas a L y eliminarlas de U'
9 Elegir aleatoriamente $2p + 2n$ instancias de U y añadirlas a U'
10 **end for**

Tri-Training

Zhou [84] en 2005 propuso una modificación sobre el algoritmo de *Co-Training* de Blum [10]. Dasgupta *et al.* [20] demostraron que cuando los requerimientos del conjunto de datos se cumplían, se podían producir un menor número de errores de generalización al maximizar la clasificación de los prototipos sin etiquetar, aunque no es aplicable a cualquier conjunto de datos.

El *Tri-Training* se define como una nueva aproximación de *Co-Training*. *Tri-Training* no necesita varias «vistas significativas» de los datos, tampoco requiere el empleo de múltiples algoritmos de aprendizaje supervisado cuyas hipótesis dividen el espacio de instancias en un conjunto de clases de equivalencia. El *Tri-Training* por tanto emplea tres clasificadores, a diferencia de los dos utilizados anteriormente, esta opción resuelve la dificultad de determinar cómo etiquetar las instancias no etiquetadas y generar la hipótesis final, lo que mejora enormemente la eficiencia del algoritmo. Junto con la capacidad de generalización resultante de la combinación de los tres clasificadores.

Algoritmo 2: *Tri-Training*

Entrada: Conjunto de entrenamiento L y U de datos etiquetados y no etiquetados, respectivamente
Entrada: *Learn* `/* algoritmo de aprendizaje */`

for $i \in \{1 \dots 3\}$ **do**
  $S_i \leftarrow BootstrapSample(L)$
  $h_i \leftarrow Learn(S_i)$
  $e'_i \leftarrow 0,5$
  $l'_i \leftarrow 0$
end for
repeat
  for $i \in \{1 \dots 3\}$ **do**
    $L_i \leftarrow \emptyset$
    $update_i \leftarrow$ **False**
    $e_i \leftarrow MeasureError\,(h_j \& h_k)\ (j,k \neq i)$
    if $e_i < e'_i$ **then**
      forall $x \in U$ **do**
        if $h_j\,(x) = h_k\,(x)\ (j,k \neq i)$ **then**
          $L_i \leftarrow L_i \cup \{(x, h_j\,(x))\}$
        end if
      end forall
      if $l'_i = 0$ **then**
        $l'_i \leftarrow \left\lfloor \frac{e_i}{e'_i - e_i} + 1 \right\rfloor$
      end if
      if $l'_i < |L_i|$ **then**
        $update_i \leftarrow$ **True**
      else if $l'_i > \frac{e_i}{e'_i - e_i}$ **then**
        $L_i \leftarrow Subsample\left(L_i, \left\lceil \frac{e'_i l'_i}{e_i} - 1 \right\rceil\right)$
        $update_i \leftarrow$ **True**
      end if
    end if
  end for
  for $i \in \{1 \dots 3\}$ **do**
    if $update_i =$ ***True*** **then**
      $h_i \leftarrow Learn\,(L \cup L_i)$
      $e'_i \leftarrow e_i$
      $l'_i \leftarrow |L_i|$
    end if
  end for
until *ningún* $h_i\,(i \in \{1 \dots 3\})$ *cambie*;

Los tres clasificadores, h_1, h_2 y h_3, son entrenados inicialmente con todo el conjunto de datos etiquetado. Seguidamente, a cualquier instancia no etiquetada, se la podrá asignar una etiqueta siempre y cuando hay al menos dos clasificadores de acuerdo con la asignación, *i.e.* $label(x) = h_i(x) = h_j(x)$, ver algoritmo 2. Puede darse el caso de que dos clasificadores acierten en la predicción y la etiqueta sea considerada correcta, pero para el tercer clasificador sea ruido, incluso en el peor caso, el aumento del ruido en el proceso de clasificación puede mitigarse si el número de instancias recién etiquetadas es significativo (bajo condiciones específicas) [84].

Debido a que *Tri-Training* no asume la existencia de clases «redundantes», se necesita un cierto grado de diversidad en los clasificadores. Esta diversidad es alcanzada mediante la manipulación del conjunto de datos etiquetado. Los clasificadores iniciales son entrenados con los datos generados mediante *bootstrap*[2] del conjunto de datos etiquetados original. Estos clasificadores son depurados en el proceso iterativo del algoritmo, produciendo la hipótesis final mediante mayoría simple.

Dado que *Tri-Training* no impone ninguna restricción al algoritmo de aprendizaje supervisado ni emplea un proceso de validación cruzada que requiera mucho tiempo de cómputo, tanto su aplicabilidad como su eficiencia demuestran ser mejores que otras versiones de *Co-Training*.

Democratic Co-Training

Zhou [83] en 2004 presentó el algoritmo *Democratic Co-Learning*. El algoritmo a diferencia de sus «homónimos», trabaja con múltiples algoritmos de aprendizaje supervisado, en lugar de múltiples clases significativas, permitiendo que se etiqueten nuevas instancias entre ellos. Debido a que diferentes algoritmos de aprendizaje poseen diferentes sesgos, seleccionar la clase más votada por la mayoría produce mejores predicciones.

El algoritmo es por tanto enfocado para el uso en casos donde no existen dos conjuntos de atributos independientes y redundantes, su pseudocódigo se encuentra disponible en los algoritmos 3 y 4.

Es por ello que *Democratic Co-Learning* utiliza un conjunto de datos etiquetados, L, uno de no etiquetados, U, y $A_1, \ldots, A_n$ para $n \geq 3$,

[2]En el campo de la estadística, se define como un método el cual consiste en la extracción de datos de muestra repetidamente con reemplazo de un conjunto de datos, con el fin de estimar un parámetro de la población.

siendo n los algoritmos de aprendizaje supervisado, ver Algoritmo 3. El algoritmo comienza entrenando los n clasificadores sobre el conjunto de datos etiquetado L; y para cada instancia x del conjunto de no etiquetados U, cada clasificador predice una etiqueta $c_i \in \mathcal{C} = \{c_1, c_2, \ldots, c_r\}$. Siendo c_k la predicción mayoritaria. Posteriormente los n clasificadores son re-entrenados con los nuevos datos etiquetados (añadidos a los que ya teníamos), este proceso se realiza de manera iterativa hasta que no se seleccionen más datos para realizar el etiquetado. Para la selección final se realizar un voto mayoritario ponderado entre los n clasificadores.

Una instancia x nunca será etiquetada por la decisión única de un clasificador, a menos que la mayoría de los clasificadores estén de acuerdo. Además se requiere que la suma de los valores medios de confianza de los clasificadores del grupo mayoritario sea mayor, que la suma de los valores medios de confianza de los clasificadores de los grupos minoritarios, siendo la confianza media de un clasificador $(l + h)/2$ para l y h definidas por el intervalo de confianza del 95 % $[l, h]$.

Finalizando con la combinación de todas las hipótesis generadas para retornar la hipótesis final. Para realizar el cálculo utiliza un sistema de votación mayoritaria de entre las posibles clases, para hilar un poco más fino, se considera también para cada clasificador su valor de confianza de la predicción.

Algoritmo 3: *Democratic Co-Learning*

Entrada: Conjunto de entrenamiento L y U de datos etiquetados y no etiquetados, respectivamente

Entrada: $A_1, \ldots, A_n$ los n algoritmos de aprendizaje supervisado

for $i = i, \ldots, n$ **do**
  $L_i \leftarrow L$
  $e_i \leftarrow 0$
end for
repeat
  for $i = 1, \ldots, n$ **do**
    Calcular las hip H_i con los datos L_i para cada clasificador A_i
  end for
  forall $x \in U$ **do**
    for $j = 1, \ldots, r$ **do** Posibles etiquetas
      $c_j \leftarrow |\{H_i | H_i(x) = j\}|$
    end for
    $k \leftarrow argmax_j\{c_j\}$
  end forall
  for $i = 1, \ldots, n$ **do** x propuestos para etiquetar
    Utilizar L para calcular el int. de conf. 95 % $[l_i, h_i]$ para H_i
    $w_i \leftarrow (l_i + h_i)/2$
    for $i = 1, \ldots, n$ **do**
      $L'_i = \emptyset$
    end for
    if $\sum_{H_j(x)=c_k} w_j > max_{c'_k \neq c_k} \sum_{H_j(x)=c'_k} w_j$ **then**
      $L'_i \leftarrow L'_i \cup \{(x, c_k)\}$, $\forall i$ tal que $H_i(x) \neq c_k$
    end if
  end for
  /* Comprobar si añadir L'_i a L mejora la precisión */
  for $i = 1, \ldots, n$ **do**
    $q_i \leftarrow |L_i| \left(1 - 2\left(\frac{e_i}{|L_i|}\right)\right)^2$ /* est. del error */
    $e'_i \leftarrow \left(1 - \frac{\sum_{i=1}^{d} l_i}{d}\right) |L'_i|$ /* est. del nuevo error */
    $q'_i \leftarrow |L_i \cup L'_i| \left(1 - \frac{2\left(e_i + e'_i\right)}{\left|L_i \cup L'_i\right|}\right)^2$ /* si L'_i es añadida */
  end for
  if $q'_i > q_i$ **then**
    $e_i \leftarrow e_i + e'_i$
  end if
until *Ningún* $L_1, \ldots, L_n$ *cambie*;

Algoritmo 4: *Democratic Co-Learning*

for $i = 1, \ldots, n$ **do**
    Calcular el int. de conf. 95 % $[l_i, h_i]$ para H_i usando L
    $w_i \leftarrow (l_i + h_i)/2$
end for
forall $x \in (L \cup U)$ **do**
    for $i = 1, \ldots, n$ **do**
        if $H_i(x)$ *predice* c_j *y* $w_i > 0{,}5$ **then**
            Asociar H_i al grupo G_j
        end if
    end for
    for $j = 1, \ldots, r$ **do**
        $\overline{C}_{G_j} \leftarrow \frac{|G_j|+0{,}5}{|G_j|+1} \times \frac{\sum_{H_i \in G_j} w_i}{|G_j|}$
    end for
end forall
Predicciones H con G_k para $k = argmax_j \left(\overline{C}_{G_j}\right)$

Self-Training basado en picos de densidad

Wu [79] en 2018 presentó un *framework* para clasificación utilizando *self-training*. En este caso a diferencia de los métodos estudiados anteriormente, se utilizan técnicas de *clustering* (agrupación) para obtener mejores resultados. Con éste método se descubre la estructura del espacio de datos, para ello se integra la densidad de los datos en el proceso de *self-training*, de manera que se entrene iterativamente un clasificador.

El proceso, ver algoritmo 5, por el cuál se consigue este nuevo clasificador «mejorado» es el siguiente:

1. Encontrar los picos de densidad de los datos para aprender la estructura subyacente de todo el espacio de datos de entrenamiento. Y se integra esta estructura en el proceso de entrenamiento iterativo de un clasificador.

2. Se entrena un clasificador con los datos etiquetados. Se clasifican los ejemplos siguientes de los ya etiquetados hasta que no haya más, se predicen y, se añaden y eliminan de los datos etiquetados y no etiquetados, respectivamente.

3. Se repite el paso anterior pero con los puntos anteriores.

Algoritmo 5: *Self-Training based on Density Peaks*

Entrada: Conjunto de entrenamiento L y U de datos etiquetados y no etiquetados, respectivamente

Salida: Clasificador entrenado

Calcular ρ_i para cada instancia $x_i \in L \cup U$
Calcular δ_i para cada instancia $x_i \in L \cup U$
Descubrir la estructura del espacio de datos haciendo que cada x_i «apunte» a su 1-NN con mayor ρ_i
Entrenar un clasificador C con L
repeat
    Seleccionar un T de U donde cada x_j es un punto «siguiente» de los $x_i \in L$ Etiquetar $x_t \in T$ con C
    $L \leftarrow L \cup T$
    $U \leftarrow U - T$
until *todos los puntos «siguientes» de* $x_i \in L$ *son seleccionados de* U;
Reentrenar C con L
repeat
    Seleccionar un T de U donde cada x_j es un punto «anterior» de los $x_i \in L$ Etiquetar $x_t \in T$ con C
    $L \leftarrow L \cup T$
    $U \leftarrow U - T$
until $size : U == 0$;
Reentrenar C con L

3.3. Minería de datos

Según IBM [22], podemos definir la minería de datos, o descubrimiento de conocimiento en los datos *Knowledge Discovery in Databases (KDD)*, como el proceso de descubrir patrones y otra información a partir de grandes conjuntos de datos.

Las técnicas de minería de datos principales se pueden dividir en función de sus propósitos principales.

1. Descripción del conjunto de datos objetivo.
2. Predicción de resultados mediante el uso de algoritmos de aprendizaje automático.

Proceso de minería de datos

El proceso de minería de datos comprende varios pasos como crear, probar y trabajar con los modelos de minería. Comienza con la recogida de los datos que van a ser tratados, y finaliza con la visualización de la información extraída de éstos. Los científicos de datos describen los datos a través de sus observaciones de patrones, asociaciones y correlaciones. A su vez se pueden clasificar y agrupar los datos utilizando métodos de clasificación y regresión.

Uno de los marcos de referencia más importantes en el proceso de minado de datos es CRISP-DM, *Cross Industry Standard Process for Data Mining*. Desarrollado por un consorcio de empresas involucradas en la minería de datos [17].

En [35] se divide el proceso de la minería de datos en 5 etapas o pasos principales: establecimiento de los objetivos y comprensión del problema, recopilación y preparación de los datos, desarrollo del modelo, aplicación del modelo y la evaluación de los resultados y despliegue en producción. Ver Figura 3.3.

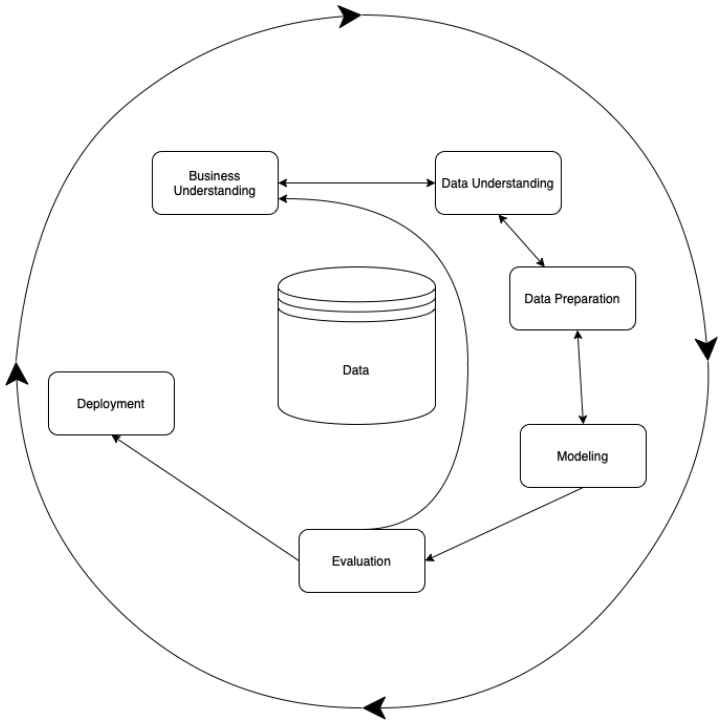


Figura 3.2: Enfoque CRISP de la minería de datos [35].

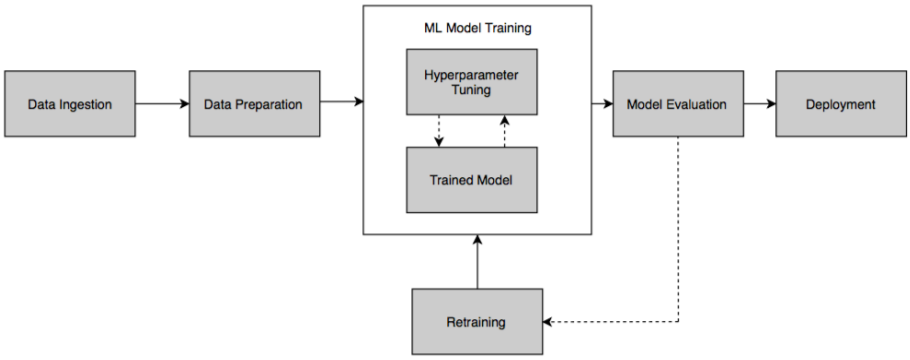


Figura 3.3: *Machine Learning Pipeline* [63].

1. **Establecer los objetivos y comprensión del problema.** La primera etapa puede resultar la más complicada del proceso. Todas las partes interesadas deben de estar presentes y de acuerdo en la definición del problema que se va tratar, esto incluye tanto a los científicos de datos como las terceras partes involucradas o interesadas. Este procedimiento ayuda a la formulación de las preguntas de los datos y los parámetros a utilizar en el proyecto. Si se trata de un proyecto

empresarial, se debe hacer un estudio o investigación adicional para comprender el contexto de la empresa.

2. **Preparación de los datos.** Con el alcance del problema definido ya se puede comenzar a identificar qué conjunto de datos será el más efectivo o representativo con el fin de comenzar a dar respuesta a las preguntas formuladas en el proceso anterior.

   Una vez se dispone de todos los datos recogidos comienza el proceso de pre-procesado de los mismos. Este proceso se basa en la limpieza de los datos con el fin de eliminar cualquier posible ruido, entendiéndose por ruido los datos duplicados, los valores perdidos y aquellos atípicos; aquellos que puedan causar problemas a la resolución del problema o generen incertidumbre. En determinados conjuntos de datos se puede hacer una reducción de dimensiones. Consiste en la reducción del número de dimensiones que poseen las instancias recogidas, con el fin de eliminar aquellas que no sean realmente representativas o significativas, este proceso reduce la complejidad computacional[3] de los cálculos posteriores. Por contrapartida hay que conocer cuáles serán los predictores con mayor relevancia en el problema para garantizar una precisión «óptima» del modelo.

3. **Desarrollo del modelo.** Según [35] el modelo es la representación abstracta de los datos y sus relaciones en un conjunto de datos concreto. Actualmente existen cientos de algoritmos que se pueden utilizar, habitualmente proceden de campos como la ciencia de datos, *machine learning*, o la estadística. Se debe tener el conocimiento suficiente para entender cómo funciona el algoritmo para poder configurar correctamente los parámetros que este va a utilizar en base a los datos y el problema de negocio que estamos resolviendo.

   En función de cómo los modelos resuelven el problema se pueden clasificar en:

   a) Regresión.

   b) Análisis de asociación.

   c) *Clustering.*

   d) Detección de anomalías.

[3] La cuantificación de la dificultad de un problema computacional en términos de recursos informáticos (como el tiempo de cálculo o la cantidad de memoria) necesarios para su solución.

El modelo debe ser creado con especial cuidado para evitar el *overfitting*, *i.e.* el modelo memoriza el conjunto de entrenamiento y no tendrá un rendimiento correcto una vez desplegado en producción. Se desea que el modelo sea lo más general posible de cara a *aprender* de los datos del conjunto de entrenamiento.

4. **Aplicación del modelo.** El momento de la aplicación del modelo es cuando de verdad se comprueba si realmente el modelo está listo para pasar al siguiente punto, en otras palabras, si es apto para ser desplegado en producción. Para ello se tienen en cuenta métricas como la calidad del modelo ante el problema, su tiempo de respuesta, etc.

5. **Evaluación de los resultados y despliegue en producción.** Es habitual que los parámetros con los que el modelo fue entrenado con el paso del tiempo dejen de ser los más interesantes, pudiendo ser comprobado el error proporcionado por el modelo con los datos de prueba. Cuando ese error sea excesivo o fuera de un margen dado se deberá de volver a entrenar el modelo, comprobar, y desplegar. De esta forma se puede comprobar como el ciclo de vida del modelo es circular.

El proceso aplicado en la minería de datos proporciona un marco de trabajo mediante el cual se permite extraer información aparentemente no trivial de grandes conjuntos de datos. Es un campo de aprendizaje constante, tanto el aplicar los conocimientos del analista para reducir las dimensiones del conjunto de datos, como una vez que se ha entrenado el modelo y puesto en producción, aprender los puntos fuertes de este y el por qué de éstos [17].

Técnicas utilizadas en la minería de datos

A continuación se presentan una serie de técnicas utilizadas en función de la naturaleza de las instancias predictoras, y de la variable de salida. Si la variable o clase de salida es continua o categórica, nos encontramos con modelos de aprendizaje supervisado, mientras que si no existe variable o clase de salida, nos encontramos con modelos de aprendizaje no supervisado [47].

1. **Reglas de asociación** (además se trata de una técnica de aprendizaje automático). Se basa en el uso de reglas básicas qutilizadas para localizar relaciones entre instancias en conjuntos de datos de gran

tamaño. Para su correcto funcionamiento deben satisfacer el soporte (nivel) mínimo especificado por el usuario y la confianza o grado de satisfacibilidad especificada en tiempo constante.

En determinadas ocasiones la generación de estas reglas es dividida en una serie de pasos:

- Para encontrar las n instancias más frecuentes, se aplica un umbral mínimo, lo cual establece la información del conjunto de datos.
- Cuando el nivel mínimo de confianza es aplicable a aquellas instancias encontradas en el paso previo, se transforman en reglas. Este paso es el que más atención requiere.

2. **Redes neuronales.** Actualmente, principalmente utilizadas en *deep learning*, simulan la interconectividad propia del cerebro humano utilizando capas de nodos. Cada nodo está compuesto por x_n entradas, w_n pesos y un sesgo o umbral, el cual al ser superado activa la neurona, pasando los datos del nodo a la siguiente neurona. El proceso es repetido a lo largo de n iteraciones pasando el mismo conjunto de entrenamiento, conocido como *epochs*.

   Las redes neuronales poseen tres ventajas en el uso de grandes conjuntos de datos: aprendizaje adaptado mediante ejemplos, robustez en el manejo de información redundante e imprecisa, y computación masiva paralela.

3. **Árboles de decisión.** Se pueden definir como particiones secuenciales de un conjunto de datos, maximizando las diferencias a una variable dependiente. Ofrecen una forma concisa de definir grupos que son consistentes en sus atributos pero que varían en términos de la variable dependiente.

   Los árboles de decisión se encuentran compuestos de nodos (variables de entrada), ramas (grupos de variables de entrada), y hojas (valores de la variable de salida). La construcción de los árboles está basada en el principio de *divide and conquer*, haciendo uso de un algoritmo de aprendizaje supervisado, se realizan divisiones sucesivas del espacio multi-variable con el objetivo de maximizar la distancia entre los grupos de cada división, *i.e.* realizar particiones discriminatorias. El proceso de división finaliza cuando todas las entradas de una rama tienen el mismo valor en el nodo hoja, dando lugar al modelo completo. Cuanto más abajo estén las variables de entrada en el árbol, menos importantes son el la clasificación de la salida.

Para evitar el *overfitting* del modelo, el árbol puede podarse eliminando las ramas con pocas instancias, o donde aquellas instancias sean poco representativas [47].

Una de las principales diferencias sobre las redes neuronales y ventaja que poseen, es la interpretabilidad que ofrecen, ofreciendo una trazabilidad de la solución propuesta.

4. **k-vecinos más cercanos. KNN (*k-nearest neighbors*)** [29, 31, 47] Las técnicas k-NN se basan en el concepto de similaridad. Permite la construcción de un método de clasificación sin hacer suposiciones sobre la forma de la función que relaciona la variable dependiente con las variables independientes.

   El objetivo es identificar de forma dinámica las k instancias en los datos de entrenamiento que son similares (vecinas) a la instancia que se quiere clasificar. La clasificación de una instancia viene dada por la observación de la clase de la vecindad, para ello se basa en los atributos de las variables. En otras palabras, cuenta el número de instancias para cada clase en la vecindad y asigna a la instancia en cuestión aquella clase que sea mayoritaria en la vecindad [19].

   Asume que todas las instancias corresponden a puntos en un espacio n-dimensional. Pudiendo ser utilizado tanto en problemas de clasificación como de regresión.

3.4. Técnicas de selección de instancias

Dentro de los conjuntos de datos nos encontramos con las instancias, también llamadas ejemplos o prototipos, son cada uno de los elementos que componen el *dataset*; en problemas reales de *machine learning* es habitual que se requiera de clasificación automática de estos datos. Este proceso se puede llevar a cabo con algoritmos de aprendizaje supervisado, Sección 3.1, con el objetivo de etiquetar la nueva información. Para poder hacerlo previamente se ha tenido que entrenado el clasificador con un conjunto de entrenamiento, T [46].

En la práctica, cualquier T dado contendrá información útil e información desechable, este último tipo de información — que en realidad son instancias — a parte de ser redundantes producen ruido, pudiendo inducir en una clasificación errónea en el proceso de aprendizaje, y posteriormente tener un modelo que no sea capaz de clasificar correctamente la nueva información.

Figura 3.4: Proceso de selección de instancias.

Es por ello que un pre-procesado o **filtrado** de las instancias pertenecientes al T original es necesario. En la Figura 3.4 se ilustra de manera gráfica el proceso de selección de instancias. Dado un conjunto de datos de entrenamiento inicial, T, el objetivo será obtener un subconjunto S, tal que $S \subseteq T$ de manera que S no contiene instancias redundantes ni «ruidosas». Además, $Acc(S) \cong Acc(T)$, donde $Acc(X)$ es la precisión, *accuracy* en inglés, del modelo entrenado con el conjunto de datos X.

En función de cómo comienzan a crear el nuevo subconjunto de datos, S, se identifican dos aproximaciones, ascendente y descendente.

- **Ascendente.** El nuevo conjunto de datos comienza estando vacío, $S = \emptyset$, y a medida que se vayan realizando iteraciones del algoritmo correspondiente, se irán añadiendo instancias a S. El principal problema que posee esta aproximación es su sensibilidad al orden, *i.e.* dada una instancia $x \subset T$, en diferentes iteraciones del mismo algoritmo de selección de instancias sobre el mismo T, puede o no estar en S. Esto se debe a la aleatoriedad con la que se presentan los datos, para asegurar esta aleatoriedad los datos se escogen de manera aleatoria de T, intrínsecamente da una mayor facilidad a las muestras iniciales a estar en S que las finales, ya que puede que ya se encuentren representadas o sean clasificadas como ruido.

  Entre las principales ventajas de esta aproximación destaca el espacio de almacenamiento requerido, puesto que se van guardando instancias y por lo tanto en un inicio es muy pequeño.

Método	**Complejidad Computacional**	**Referencia**
Edición de Wilson (ENN)	$O(n^2)$	[77]
Condensado de Hart (CNN)	$O(n^3)$	[32]
Condensado Reducido (RNN)	$O(n^3)$	[27]
Iterative Case Filtering (ICF)	$O(n^2)$	[13]
Subconjunto Selectivo Modificado (MSS)	$O(n^2)$	[7]
Drecremental Reduction Optimization Procedure (DROP)	$O(n^2)$	[76]

Tabla 3.1: Algunos métodos de selección de instancias.

- **Descendente.** El nuevo conjunto de datos comienza siendo el conjunto de entrenamiento al completo, $S = T$, y a medida que se vayan realizando las iteraciones del algoritmo correspondiente, se irán eliminando instancias de S. Esta aproximación es mucho más costosa computacionalmente, para cada instancia que debe decidir si eliminar o no debe comprobar todo el subconjunto S, pero en contraposición consigue reducir más que la aproximación ascendente, el conjunto de entrenamiento T.

Junto con esta diferenciación, en función de la aproximación de selección de instancias se pueden distinguir dos agrupaciones.

- ***Wrapper.*** El criterio de selección se basa en la precisión del clasificador. Aquellas instancias que no contribuyen a la mejora del clasificador se quedan fuera de S. Este trabajo está centrado en este criterio de selección.

- ***Filter.*** El criterio de selección utiliza una función, $f(x, y)$, para realizar la selección, no se basa en un clasificador concreto.

En la Tabla 3.1 se aprecian aquellos algoritmos implementados en primera instancia para la reducción de instancias dentro de T, el objetivo de todos ellos es que el subconjunto generado, S, sea capaz de clasificar correctamente T en su totalidad prácticamente.

Algoritmos de selección de instancias

Existen multitud de algoritmos a día de hoy que son capaces de reducir el número de instancias de T, en este trabajo se van a comentar los pertenecientes a la Tabla 3.1. Cada uno de ellos tiene sus ventajas y sus desventajas como cabe esperar, en la literatura no apreciamos un «este es mejor que aquel» o similar.

Algoritmo de edición de Wilson

Wilson [77] (1972) publicó la regla del vecino más cercano editado, *ENN*. Los problemas de clasificación de instancias en función de una etiqueta, se caracterizan por:

- Hay una instancia a ser clasificada.
- Existe un S el cual posee instancias con la misma distribución que la instancia a clasificar, pudiendo ser comparables las instancias del conjunto con la que estamos analizando.
- No existe información adicional del conjunto.
- Existe una distancia medible entre instancias.

Algoritmo 6: Algoritmo de edición de Wilson, *ENN*

Entrada: Conjunto de entrenamiento $X = \{(x_1, y_1,) \dots (x_n, y_n)\}$, k número de vecinos.
Salida: Conjunto editado $S \subset X$

```
1 S ← X
2 forall x ∈ S do
3     Encontrar x.N_{1...k+1}, los k + 1 vecinos más cercanos de
        x ∈ X − {x}
4     if δ_{k−NN}(x_i) ≠ θ_i then
5         Eliminar x de S
6     end if
7 end forall
8 return S
```

Con todas estas premisas Wilson propone un algoritmo basado en clasificación incorrecta en función de sus vecinos más cercanos. Cuando una instancia resulta mal clasificada por sus k vecinos más cercanos, k-NN,

esa instancia es descartada. Finalmente obtendremos como resultado un conjunto S con las instancias correctamente clasificadas por sus vecinos.

Suponiendo que sea X un conjunto de N instancias y M posibles clases y, sea k el número de vecinos cercanos, el algoritmo de Wilson se puede formular de la siguiente manera:

El algoritmo de edición de Wilson, ver algoritmo 6, posee una complejidad computacional de $O(n^2)$. Una de las ventajas de este algoritmo es su forma de crear el subconjunto S, ya que al ser descendente las primeras iteraciones serán lentas — dependiendo del tamaño de T lógicamente — pero las finales serán considerablemente más rápidas (siempre y cuando haya un nivel significativo de ruido).

Algoritmo Condensado de Hart

Hart [32] en 1968 propuso la que se considera la primera regla formal de condensador para NN. El algoritmo de condensado de Hart, *CNN — Condensed Nearest Neighbor*. Está basado en técnicas de consistencia y reducción. Sea $X \neq \emptyset$ y $S \subseteq X$, podremos decir que el subconjunto S es consistente respecto al conjunto X si al utilizar a S como conjunto de aprendizaje, se puede clasificar correctamente a todo el conjunto X.

El algoritmo de Hart es una técnica ascendente, a partir de las primeras instancias que se añadan a S se clasificarán y añadirán, o no, a S. Consiste en encontrar entre todas las instancias de T un subconjunto S tal que cada instancia de T sea más cercano a las instancias en S de su misma clase que a las instancias de otras clases, permitiendo utilizar S como conjunto de clasificación de T. Para el correcto funcionamiento se asume que el conjunto T es consistente, no posee dos instancias idénticas con pertenencia a diferentes clases.

El algoritmo propuesto por Hart [32], ver algoritmo 7, posee una complejidad computacional de $O(n^2)$. El conjunto obtenido a partir de T, *i.e.* S, operando con grandes conjuntos de datos demuestra poseer un tamaño considerablemente menor respecto a T.

Si bien es una técnica utilizada por su efectividad, posee una serie de puntos negativos a su vez.

- Sensibilidad ante el ruido. Un objeto ruidoso no será correctamente clasificado por sus vecinos. Estas muestran no se eliminarán del conjunto solución S, por lo que no desaparecerán.
- S no tiene porqué ser el menor conjunto de T. Diferentes ejecuciones del algoritmo sobre el mismo T pueden dar diferentes conjuntos solución S. Esto se debe al orden aleatorio por el cual se seleccionan las instancias. Por definición del propio algoritmo se asume que no se va a alcanzar de forma general el subconjunto de tamaño mínimo que cumpla con las características especificadas.

Algoritmo 7: Algoritmo Condensado de Hart, *CNN*.

Entrada: Conjunto de entrenamiento X
Salida: Conjunto editado $S \subset X$

```
1 S ← {x_1}
2 forall x ∈ S do
3     if x no se clasifica correctamente usando S then
4         Añadir x a S
5         Reiniciar
6     end if
7 end forall
8 return S
```

Algoritmo Condensado Reducido

Gates [27] en 1972 propuso el algoritmo del conjunto reducido, basado en las reglas *NN*. El algoritmo propuesto es una modificación del algoritmo *CNN*, ver algoritmo 7. No es una nueva regla de decisión puesto que se sigue eligiendo la clase del vecino más cercano para la clasificación. El algoritmo se basa en un procedimiento para seleccionar el subconjunto T_{CNN}, el cual debe comportarse igual de bien que T_{NN} ante clasificaciones de instancias desconocidas. Como se puede apreciar en el propio algoritmo 8, puede existir una disminución del rendimiento, a cambio se obtiene una mejora de la eficiencia para el algoritmo de clasificación que posteriormente se utilice, tanto en la cantidad de memoria utilizada como en el tiempo de computación.

De igual manera que en *CNN*, posee la problemática de la minimalidad, si bien el conjunto resultante $S \subset T$ será consistente, no se puede asegurar que sea mínimo; y diferentes ejecuciones del algoritmo sobre el mismo conjunto de datos T, pueden obtener diferentes subconjuntos solución S.

Algoritmo 8: Algoritmo Condensado Reducido, *RNN*.

Entrada: Conjunto de entrenamiento X
Salida: Conjunto editado $S \subset X$

```
1  S ← {x_1}
2  forall x ∈ S do
3      if x no se clasifica correctamente usando S then
4          Añadir x a S
5          Reiniciar
6      end if
7  end forall
8  forall x ∈ S do
9      Eliminar x de S
10     if ∃x_i incorrectamente clasificada usando S then
11         Añadir x a S
12     end if
13 end forall
14 return S
```

Algoritmo *Iterative Case Filtering*

Brighton [13] en 2002 propuso el algoritmo iterativo de filtrado, *ICF*, bajo la premisa de predecir la clase de una instancia con la misma precisión, o mayor si fuera el caso, que el T original. Uno de los objetivos principales del propio algoritmo es mantener en el subconjunto S únicamente aquellas instancias que sean críticas para la decisión.

ICF utiliza una estrategia decremental, *i.e.* de manera iterativa va borrando instancias del conjunto de datos inicial, inicialmente realiza un filtrado y posteriormente la reducción [13] Al reducir el tamaño de T los tiempos de respuesta para el proceso de clasificación mejorarán, ya que se examinarán un menor número de instancias para realizar la clasificación; por contrapartida al eliminar muestras que se creen que son dañinas para el proceso, se pueden estar eliminando algunas que sean clave y por lo tanto teniendo una degradación de la calidad del clasificador.

ICF se basa en dos categorías para la decisión de si una instancia debe permanecer o no en S, ver algoritmo 9, estas son *Coverage* y *Reachable*. Definidas de la siguiente manera para el caso base $\mathcal{CB} = \{x_1, x_2, ..., x_n\}$.

$$Coverage(x) = \{x' \in \mathcal{CB} : Adaptable(x, x')\}$$
$$Reachable(x) = \{x' \in \mathcal{CB} : Adaptable(x', x)\}$$

Una instancia x podrá estar en el conjunto adaptable de x' sí y solo si x es una instancia relevante para la solución de x', *i.e.* $x \in k-NN(x)$. La problemática surge en el momento en el cual una instancia con clase diferente no permite la correcta clasificación de x', es por ello que el vecindario de x' viene definido por todas las instancias antes de la primera instancia de diferente clase, utilizando *sets* descritos por Wilson y Martinez. La propiedad *Reachable* no está fijada desde el principio del algoritmo, sino que es dinámica siendo fijada cada vez en función de la instancia de clase diferente más cercana. El criterio seguido para realizar la eliminación de una muestras es: si el *set* formado por el $reachable(x)$ es mayor que el $coverage(x)$, *i.e.* una instancia x es eliminada cuando más instancias pueden resolver x que las que x puede resolver por sí misma.

Algoritmo 9: *Iterative Case Filtering, ICF.*

Entrada: Conjunto de entrenamiento X
Salida: Conjunto editado $T \subset X$

```
   /* Filtro de ruido en función de la regla k-NN        */
 1 T ← ENN(X, k)
 2 repeat
 3     forall x ∈ T do
 4         Calcular coverage(x)
 5         Calcular reachable(x)
 6     end forall
 7     progress ← false
 8     forall x ∈ T do
 9         if |reachable(x)| > |coverage(x)| then
10             Marcar x para eliminar
11             progress ← true
12         end if
13     end forall
14     forall x ∈ T do
15         if x marcada para eliminar then
16             Eliminar x de T
17         end if
18     end forall
19 until no progress;
20 return T
```

Algoritmo Subconjunto Selectivo
Ritter [56] 1975 propone un algoritmo capaz de satisfacer la condición de consistencia del algoritmo condensado de Hart, ver algoritmo 7; para ello introduce una condición más «fuerte» de consistencia, el objetivo es encontrar aquellas instancias de un orden independiente.

Un subconjunto S del conjunto de entrenamiento T, es un Subconjunto Selectivo, SS, si SS cumple las siguientes condiciones:

1. El subconjunto S debe ser consistente.
2. Todas las instancias tiene que ser más cercanas a un vecino selectivo de la misma clase que a cualquier otra instancia de otra clase.
3. No puede haber ningún subconjunto S' que satisfaga las condiciones 1 y 2, y que al mismo tiempo contenga un menor número de instancias que el Subconjunto Selectivo, SS.

La segunda condición es la diferencia principal entre el algoritmo de Hart y el subconjunto calculado con el algoritmo del subconjunto selectivo. De forma que se puede re-formular el punto número dos de la siguiente manera:

Todas las instancias del conjunto de entrenamiento T deben de ser más cercanos a un vecino condensado — miembro del subconjunto condensado $\mathcal{CS}$ — de la misma clase que a cualquier otra instancia de otra clase diferente del $\mathcal{CS}$.

Pudiendo verse como una reformulación del punto nº 1. Además, el punto nº 2 para el subconjunto selectivo permite un subconjunto de menor tamaño, eliminando la necesidad de calcular todas las permutaciones de las instancias en T. Con unos criterios más específicos que los que se encuentran en el algoritmo condensado, el subconjunto selectivo resultante no tiene porqué ser mínimamente consistente. Asimismo, de forma general, no será un subconjunto reducido del S producido por *CNN*.

Algoritmo Subconjunto Selectivo Modificado
Barandela [7] 2005 propuso el algoritmo de subconjunto selectivo modificado, *MSS*, como su propio nombre indica, se trata de una modificación del algoritmo Subconjunto Selectivo, SS. Debido a que no se puede garantizar que el algoritmo de Ritter [56] devuelva un subconjunto, S, el cual sea

mínimo y consistente, habiendo sido categorizado como un problema NP-Completo [75], por lo que *MSS* pretende conseguir el subconjunto mínimo consistente mediante el uso de la propiedad selectiva.

La aproximación realizada por Barandela *et al.* modifica la condición nº 3 anteriormente propuesta, mientras que las nº 1 y 2 no son modificadas. De forma que la definición nº 3 queda formulada de la siguiente manera:

El Subconjunto Selectivo Modificado, MSS, se define como el subconjunto del conjunto de entrenamiento TS, el cual $\forall x_i \in TS$ aquella instancia de Y_i qiue es más cercano a otra clase que a la de x_i, i.e. el más cercano a su enemigo más cercano.

El objetivo principal de esta modificación es reforzar la condición que debe cumplir el subconjunto reducido para maximizar la aproximación a la frontera de decisión. Quedando definido el algoritmo, ver algoritmo 10, como una alternativa eficiente al algoritmo propuesto por Ritter *et al.*, siendo capaz de seleccionar mejores instancias (más cercanas a la frontera de decisión).

El criterio que sigue *MSS* para determinar la frontera de decisión es la distancia al enemigo más cercano. Con esta medida se puede definir el mejor subconjunto selectivo como aquel que contiene el mejor vecino relacionado para cada instancia en el TS. (Mejor $\iff$ Menor distancia a su enemigo más cercano).

Algoritmo 10: *Modified Selective Subset, MSS.*

Entrada: Conjunto de entrenamiento X
Salida: Conjunto editado $S \subset X$

```
1  S ← ∅
2  Ordenar las instancias {x_i}_{i=1}^n en función de D_i a su enemigo más
     cercano
3  for i = 1 hasta n do
4      add ← false
5      for j = i hasta n do
6          if x_j ∈ T ⋀ d(x_i, x_j) < D_j then
7              Eliminar x_j de T
8              add ← true
9          end if
10     end for
11     if add then
12         Añadir x_i a S
13     end if
14     if T = ∅ then
15         return S
16     end if
17 end for
```

Decremental Reduction Optimization Procedure

Bajo el nombre de *Decremental Reduction Optimization Procedure, DROP* [76] nos encontramos con hasta 5 aproximaciones para la resolución del mismo problema. Todas ellas seleccionan prototipos en función de *socio* y *asociado*. Definidos como:

- Socio. Sea $X \neq \emptyset$, el socio de un objeto P el cual pertenece al conjunto X, es aquél objeto que tiene a P como uno de sus k vecinos más cercanos.
- Asociado. Todo prototipo que contenga a P como a uno de sus k vecinos más cercanos, será denominado asociado de P, denotado con la expresión $P.A_{1,\ldots,a}$ donde a es el número de asociados de P.

Volviendo sobre los conceptos de *coverage* y *reachable* vistos en el algoritmo *Iterative Case Filtering*, (ICF), podemos relacionar todos los conceptos de la siguiente manera: los asociados de un prototipo P se pueden

ver como el conjunto formado por el *reachable* de P, y con el *coverage* se pueden obtener los socios del prototipo P.

Todas las aproximaciones de $DROP\{1 \dots 5\}$ se basan en el mismo algoritmo [76] pero con modificaciones en las técnicas de pre-procesado de los prototipos. Sus diferencias son:

- *DROP1*. Eliminará un objeto P de S si sus socios en S se clasifican correctamente sin P, *i.e.* la ausencia de P no impide la correcta clasificación del resto de prototipos.

- *DROP2*. Eliminará un objeto P de S si los socios que tiene P en TS se clasifican correctamente sin P, *i.e.*, verificará el efecto que causa esta eliminación sobre la muestra original. Previo al proceso de eliminación de objetos P, ordena los prototipos a tratar en orden a su enemigo más cercano, permitiendo que los primeros prototipos que se van a tratar serán aquellos más alejados de las fronteras de decisión, ergo las más prescindibles.

- *DROP3*. Lo primero de todo realiza un filtrado de ruido, para ello aplica la edición de Wilson, ver algoritmo 6. Seguidamente aplica el algoritmo *DROP2*, ver algoritmo 11

- *DROP4*. Aplica un filtro de ruido diferente, en este casos consistirá en eliminar un prototipo solo si su eliminación no provoca que alguna otra instancia sea mal clasificada.

- *DROP5*. Modificación sobre *DROP2*, en este algoritmo el proceso de eliminación de objetos comienza por aquellos más cercanos a sus enemigos.

El concepto *with* permite recoger el número de asociados correctamente clasificados teniendo en cuenta un prototipo x como vecino, en cambio *without* tiene en cuenta el número de asociados correctamente clasificados los cuales no tienen el prototipo x como vecino. En los casos en los que $without > with$ se está tratando con un prototipo el cual no aporta información sobre el conjunto S permitiendo su eliminación.

Algoritmo 11: *Decremental Reduction Optimization Procedure 3, DROP3.*

Entrada: Conjunto de entrenamiento X y número de vecinos más cercanos a considerar, k
Salida: Conjunto editado $S \subset X$

```
1  S ← Filtrado de ruido(X)
2  Ordenar las instancias de S por la distancia a su enemigo más
     próximo                                   /* de mayor a menor */
3  forall x ∈ S do
4      Encontrar los x.N_{1...k+1}, los k+1 vecinos más cercanos de x en S
5      Añadir x a la lista de asociados de cada uno de sus vecinos
6  end forall
7  forall x ∈ S do
8      with ← número de asociados de x clasificados correctamente con
         x como vecino
9      without ← número de asociados de x clasificados correctamente
         sin x como vecino
10     if without ≥ with then
11         Eliminar x de S
12         forall Asociado a de x do
13             Eliminar x de la lista de vecinos de a
14             Encontrar nuevo vecino para a
15             Añadir a en la lista de asociados del nuevo vecino
16         end forall
17     end if
18 end forall
19 return S
```

3.5. Función distancia entre instancias

Una función distancia proporciona información sobre la proximidad entre dos instancias en función de todos sus parámetros. Si la distancia que separa dos instancias es cero, ambas instancias son idénticas. Se tiende a trabajar con conjuntos de datos normalizados, *i.e.* todos los datos son ajustados a una escala común, independientemente de la escala en la que hayan sido medidos, para evitar que atributos/características con mucha varianza puedan «despistar» a los algoritmos.

Existen multitud de métricas para calcular la distancia, pudiendo variar la distancia calculada en función de cuál se aplique. Se van a comentar las más representativas.

- **Distancia de Minkowski.** La distancia de Minkowski es una métrica en el espacio vectorial normalizado. Es una métrica que se puede modificar con facilidad para calcular la distancia entre dos instancias de diferentes maneras.

  1. $p = 1$, cálculo de la distancia de Manhattan.
  2. $p = 2$, cálculo de la distancia Euclídea.
  3. $p = \infty$, cálculo de la distancia de Chebyshov.

  Su fórmula es la siguiente:

$$\mathbb{D}(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

- **Distancia de Manhattan** o distancia del taxista. Es una métrica en un espacio vectorial normalizado, calculándose como la suma de los n segmentos verticales u horizontales que unen dos puntos.

  Su fórmula es la siguiente:

$$\mathbb{D}(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$

- **Distancia euclidiana** o norma L2 o distancia L2. Es la distancia en línea recta entre dos puntos de datos en el espacio euclidiano.

  Su fórmula normalizada es la siguiente:

$$\mathbb{D}(x, y) = \sqrt{\sum_{i=1}^{d} \frac{(x_i - y_i)^2}{\sigma_i^2}}$$

- **Distancia de Chebyshov** o distancia del tablero de ajedrez. La distancia entre dos puntos es la mayor de sus diferencias a lo largo de cualquiera de sus dimensiones coordenadas.

  Su fórmula es la siguiente:

$$\mathbb{D}(x, y) = \max_i \left(|x_i - y_i| \right)$$

Técnicas y herramientas

En este capítulo se van a comentar las técnicas y herramientas utilizadas a lo largo del desarrollo del proyecto *software*.

4.1. Técnicas

Metodología SCRUM

Scrum es un marco de trabajo que permite el trabajo colaborativo en equipos. Permite que los equipos que trabajan en proyectos con esta metodología se organicen por sí mismos, siendo ellos los que deciden cómo afrontar los problemas que van surgiendo.

Según [15], el modelo *Scrum* se basa en tres componentes principales: roles, procesos y artefactos. El *Scrum Master* es el puesto asumido por el director o gerente del proyecto, o en algunos casos el líder del equipo. Esta figura representa los valores y principios por los que se rige la metodología de *scrum*, manteniendo los valores y buenas prácticas, así como resolviendo los impedimentos que vayan surgiendo a lo largo del desarrollo del proyecto. Habitualmente los equipos están compuestos por entre cinco y diez personas que trabajan en el proyecto a tiempo completo. Siendo este equipo independiente y flexible en cuanto a jerarquía interna, no siendo representado el papel del «jefe» dentro de este por la misma persona siempre. Esto genera que el papel cambie en función de las necesidades del propio proyecto, la configuración del equipo cambia únicamente entre iteraciones, o *sprints*, no dentro de los mismos.

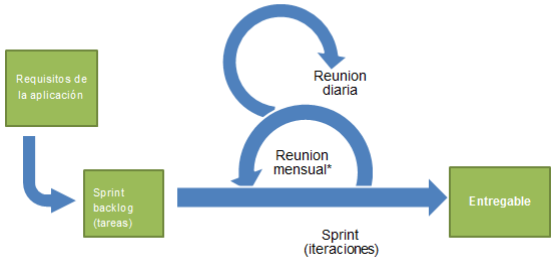


Figura 4.1: Metodología *scrum* [3].

Sprints

Los *sprints* son periodos breves de **tiempo fijo** en el que el equipo trabaja para completar una cantidad de trabajo pre-establecida. Si bien muchas guías asocian los *sprints* a la metodología ágil, asociando la metodología ágil y la metodología seguida en *scrum* como si fueran lo mismo, cuando no lo son. La metodología ágil constituye una serie de principios, y la metodología *scrum* es un marco de trabajo con la única finalidad de conseguir resultados.

A pesar de las similitudes los *sprints* poseen un objetivo subyacente, entregar con frecuencia *software* de trabajo.

Sprint meetings

Dentro de la metodología *scrum* existen diferentes reuniones que favorecen la agilidad del proyecto y que todo el mundo sepa lo que tiene que hacer en cada momento.

- ***Sprint planning meeting.*** Esta reunión puede tener una duración de hasta de un día completo de trabajo. En ella deben de estar presentes todas las partes del proyecto, *i.e.* el *Scrum Master*, el equipo de desarrollo, y el *product owner*. Poseen dos partes, en la primera de ellas se define el *product backlog*, requerimientos del proyecto y se definen los objetivos para el *sprint* que comienza, *i.e.* lo que se espera "construir" o completar en el *sprint*. En la segunda parte de la reunión se trabaja en el *sprint backlog*, las tareas que se van a seguir en el *sprint* para completar el objetivo de éste.
- ***Daily meeting.*** Debido a que los requerimientos del proyecto no se pueden variar durante la vida de un *sprint*, existen las reuniones diarias

que son organizadas por el *Scrum Master* en las que se comenta el trabajo del día previo, lo que se espera de ese día y qué está retrasando o impidiendo a un individuo el proseguir con sus tareas, esta reunión no debe tener una duración de más de quince minutos y se debe realizar "de pie". No es una reunión para ver quién retrasa el proyecto sino para ayudar a quién lo necesite entre todos los miembros del equipo y permitir esa agilidad.

- ***Sprint review meeting.*** Reunión fijada al final de cada *sprint* en la cual se hace una puesta en conocimiento de lo que se ha realizado en ese *sprint*, siempre que se pueda se hará una demostración funcional en lugar de una presentación al *product owner*. Esta reunión tiene un carácter informal.

Artifacts

Uno de los componentes más importantes de cara a la metodología *scrum* son los artefactos, o *artifacts* por su nombre en inglés. Éstos incluyen el *product backlog*, el *sprint backlog* y los *burn down charts*.

- ***Product backlog.*** Lista de trabajo ordenada por las prioridades para el equipo de desarrollo. Es generada a partir de las reuniones de planificación de los *sprints*, contiene los requisitos. Se encuentra actualizado y clasificado en función de la periodicidad asignada a las tareas, pudiendo ser de corto o largo plazo. Aquellas tareas que se deban resolver a corto plazo deberán estar perfectaemnte descritas antes de asignarlas esta periodicidad, implicnddo que se han diseñado las historias de usuario completas así como el equipo de desarrollo ha establecido las estimaciones correspondientes. Los elementos a largo plazo pueden ser abstractos u opacos, conviene que estén estimados en la medida de lo posible para poder tener en cuenta el tiempo que llevará desarrollarla.

  Los propietarios del producto dictan la prioridad de los elementos de trabajo en el *product backlog*, mientras que el equipo de desarrollo dicta la velocidad a la que se trabaja en *backlog* [49].

  La estimación es una parte muy importante ya que es lo que permitirá al equipo de desarrollo mantener el ánimo y el trabajo al ritmo deseado. La estimación es realizada en la *sprint planning meeting*, en la que se estima para cada tarea/producto del *product backlog*. No se busca tener un resultado exacto del tiempo que va a llevar al equipo completar esa

tarea, sino es una previsión. Para realizar correctamente la estimación se debe tener en cuenta el tamaño y la categoría de la tarea, los puntos de historia que se le van a asignar, así como el número de horas y días que van a ser necesarias para completar la tarea.

- ***Sprint backlog.*** Lista de tareas extraídas del *product backlog* que se han acordado desarrollarse a lo largo de un *sprint*. Este *backlog* es seleccionado por el propio equipo de desarrollo, para ello seleccionan una tarea del *product backlog* y se divide en tareas de menor tamaño y abordables. Aquellas tareas de menor tamaño que el equipo no haya sido capaz de desarrollar previo a la finalización del *sprint* quedarán almacenadas para próximos *sprints* en el *sprint backlog*.

Actores, roles y responsabilidades

Dentro de un equipo que sigue la metodología *scrum* encontramos diferentes actores, como ya se ha comentado el equipo de desarrollo suele estar compuesto por entre cinco y diez personas, además del *Scrum Master* y el *Product Owner* [57].

- ***Product Owner.*** Encargado de optimizar y maximizar el valor del producto, es la persona encargada de gestionar las prioridades del *product backlog*. Una de sus principales tareas es la de intermediario con los *stakeholders*, partes interesadas, del proyecto; junto con recoger los requerimientos de los clientes. Es habitual que esta figura sea representante del negocio, con lo que aumenta su valor.

  Para cada *sprint* debe de marcar el objetivo de éste de manera clara y acordada con el equipo de desarrollo, lo cual hará que el producto vaya incrementando constantemente su valor. Para que todo fluya como debe, esta figura tiene que tener el "poder" de tomar decisiones que afecten al producto.

- ***Scrum Master.*** Figura con dos responsabilidades, gestionar el proceso *scrum* y ayudar a eliminar impedimentos que puedan afectar a la entrega del producto.

  1. Gestionar el proceso *scrum*. Su función es asegurarse de que el proceso se lleva a cabo correctamente, facilitando la ejecución de éste y sus mecánicas. Consiguiendo que la metodología sea una fuente de generación de valor.

2. Eliminar impedimentos. Eliminar los problemas que vayan surgiendo a lo largo de los *sprints* con el fin de mantener el ritmo de trabajo dentro de los equipos de desarrollo para poder entregar valor, manteniendo la integridad de la metodología.

- **Equipo de desarrollo.** Formado por entre cinco y diez personas encargados del desarrollo del producto, organizados de forma autónoma para conseguir entregar las tareas del *product backlog* asignadas al *sprint* correspondiente. Para que funcione correctamente la metodología todos los integrantes deben de conocer su rol dentro del equipo, internamente se pueden gestionar como el equipo considere, pero de cara «hacia fuera» son un equipo con una responsabilidad.

Python

Según [71], Python es un lenguaje de programación multiplataforma y multiparadigma[4], entre sus principales características se encuentran la legibilidad y limpieza del código. Python es *open source software* por lo que su uso es gratuito [24].

Python fue desarrollado al principio de la década de 1990 por Guido van Rossum en el Stichting Mathematish Centrum, CWI, en el Reino de los Países Bajos. Implementado como el sucesor del lenguaje ABC. Actualmente el lenguaje sigue siendo desarrollado por Guido, junto con contribuciones de muchos otros. El lenguaje está fuertemente influenciado por otros como: ABC, Ada, ALGOL 68, APL, C, C++, CLU, Dylan, Haskell, Icon, Java, Lips, Modula-3, Perl, Standard ML.

El principal objetivo de Python es la automatización de procesos con el fin de minimizar tanto complicaciones como tiempo de desarrollo. Debido a estas dos características, a día de hoy Python es el lenguaje de programación más utilizado para desarrollo de todo tipo de aplicaciones, tanto es así que, según las últimas estadísticas recuperadas por PYPL, ver [14], además es el lenguaje de programación sobre el que más tutoriales se buscan en el top buscadores (Google, Bing, DuckDuckGo,...). Se aplica en campos tales como:

- Ciencia de datos. Uso de las bibliotecas de Python desarrolldas para el análisis y visualización de datos.

[4]Debido a que soporta parcialmente la orientación a objetos, la programación imperativa, y la programación funcional.

- Aprendizaje automático. Dado que Python es un lenguaje de programación tan estable, flexible y sencillo, es perfecto para varios proyectos de aprendizaje automático (ML) e inteligencia artificial (AI). De hecho, Python es uno de los lenguajes favoritos entre los científicos de datos, y hay muchas bibliotecas y paquetes de aprendizaje automático e IA disponibles en Python.
- Desarrollo web. Utilizado principalmente en el *back-end* de las aplicaciones web, tanto en la codificación de las funcionalidades, de manera similar a lo que JavaScript permitiría hacer; como encargándose de la conexión con las diferentes bases de datos que puedan estar en producción.
- Visión artificial y procesamiento de imágenes.
- Desarrollo de videojuegos.
- Medicina y bioinformática.
- Astronomía.

PEP8

La abreviación PEP hace referencia a *Python Enterprise Proposal* [2]. La escritura de código con la lógica adecuada en un factor clave independientemente del lenguaje de programación en el que se esté trabajando, pero hay muchos más factores que influyen en la calidad del código. El estilo de codificación del programador hace que el código sea mucho más fiable, se debe de tener en cuenta que Python sigue estrictamente la forma de orden y formato de la cadena, *i.e.* tal y como se escribe el código se procesa para su compilación.

PEP8 es un documento el cual proporciona varias directrices para escribir la parte legible en Python. Fue escrito oficialmente en 2001 por Guido van Rossum, Barry Warsaw y Nick Coghlan, con el objetivo principal de mejorar la legibilidad y consistencia del código. Disponible para consulta [59].

PEP8 mejora la legibilidad del código, pero, ¿a qué se debe que la legibilidad sea tan importante en Python? «*Code is much more often than it is written.*» [61]. Un fragmento de código puede ser escrito en cuestión de minutos o unas pocas horas, pero una vez escrito no se reescribirá nunca, pero seguramente sí que será leído un número indefinido de veces. Es en ese preciso momento cuando se debe tener una idea del por qué de esa línea de

código. El código debe reflejar el significado de cada línea, de ahí el motivo de que la legibilidad sea tan importante.

Investigación

Ranking medio

En el campo de la estadística, los *rankings* permiten la transformación de datos en función de su posición cuando el conjunto de datos es ordenado.

El *ranking* medio se calcula de la siguiente manera: para una serie de conjuntos de datos los cuales han sido utilizado por diferentes clasificadores, cada uno de estos habrá reportado unos resultados para cada uno de los conjuntos de datos y sobre éstos últimos se puede obtener un *ranking*, el *ranking* medio consiste en calcular la media de los *rankings* reportados para cada conjuntos de datos por cada clasificador.

Test estadístico

Mecanismo para tomar decisiones cuantitativas sobre un proceso o una serie de estos. El objetivo es determinar si hay suficientes pruebas para «rechazar» una hipótesis sobre el proceso. La conjetura se denomina hipótesis nula. No rechazarla puede ser un resultado si se quiere seguir actuando como si se «creyera» que la hipótesis nula es cierta. O puede ser un resultado decepcionante, que posiblemente indique auq aún no se poseen suficientes datos para «demostrar» algo rechazando la hipótesis nula [42] [45].

4.2. Herramientas

UBUMLaaS

UBUMLaaS surge como una plataforma de *Machine Learning as a Service* basada en los métodos desarrollados tanto por el grupo de investigación ADMIRABLE[5]. Junto con los desarrollados por BEST-AI[6].

[5] *Advanced Data MIning Research And (Business intelligence | Bioinformatics | Big data) LEarning*. El objetivo principal del grupo de investigación es el desarrollo de nuevos algoritmos de *ensemble* y la aplicación de técnicas de minería de datos y *pattern matching* a diversos campos como la bioinformática, la clasificación de series temporales y el análisis de datos de alta dimensión [1]

[6] El Grupo de Investigación BEST-AI (Biología, Educación y Salud con Tecnologías Avanzadas Informáticas) de la Universidad de Burgos, centra su actividad investigadora

El proyecto permite a terceros, registrados en la plataforma, hacer uso de técnicas de aprendizaje automático en la nube. En una primera instancia fue desarrollado por miembros del grupo ADMIRABLE.

La plataforma, previa a este Trabajo de Fin de Grado, proporciona únicamente soporte a algoritmos de aprendizaje supervisado, estos algoritmos provienen de diversas bibliotecas como son Weka, Meka, Scikit-Learn o los propios algoritmos desarrollados por el grupo ADMIRABLE. Habiendo siendo paralizado su soporte a mediados de 2019.

Página web de la herramienta: `http://ubumlaas.eu.ngrok.io`

Weka

Desarrollado por la Universidad de Waikato [78] es un *open source software* el cual provee de herramientas para el pre-procesado de datos, implementación de algoritmos de *Machine Learning*, y herramientas de visualización, de tal manera que permite desarrollar técnicas de aprendizaje automático y aplicarlas a problemas de minería de datos.

En la figura 4.2 se observa un diagrama con todas las funcionalidades que ofrece Weka. Para poder utilizar un conjunto de datos en *Machine Learning*, ver sección 3.1, hay que hacerlos aptos, esto se consigue con el primer paso, el pre-procesado, para ello Weka proporciona una serie de herramientas que permiten limpiar los valores nulos, campos irrelevantes, etcétera. Seguidamente se guardan los datos que han sido pre-procesados en almacenamiento local para aplicar algoritmos de *Machine Learning*[7].

En función de qué tipo de modelo de ML se desee aplicar, se selecciona una opción de entre: clasificación, agrupamiento, asociación, o selección de atributos. Weka proporciona una serie de algoritmos para cada modelo, permitiendo su completa parametrización individualizada, de forma que se pueden realizar los experimentos al gusto del investigador.

Finalmente Weka proporciona un resultado estadístico del procesamiento del modelo, junto con ello proporciona una herramienta de

en el desarrollo de nuevos algoritmos de minería de datos e inteligencia artificial y en su aplicación a problemas biológicos, bioinformáticos, sanitarios, medioambientales o educativos.

[7]Esta es una de las principales desventajas de Weka, almacena el conjunto de datos en su totalidad en memoria del sistema, limitando el número de instancias que puede tener un conjunto de datos.

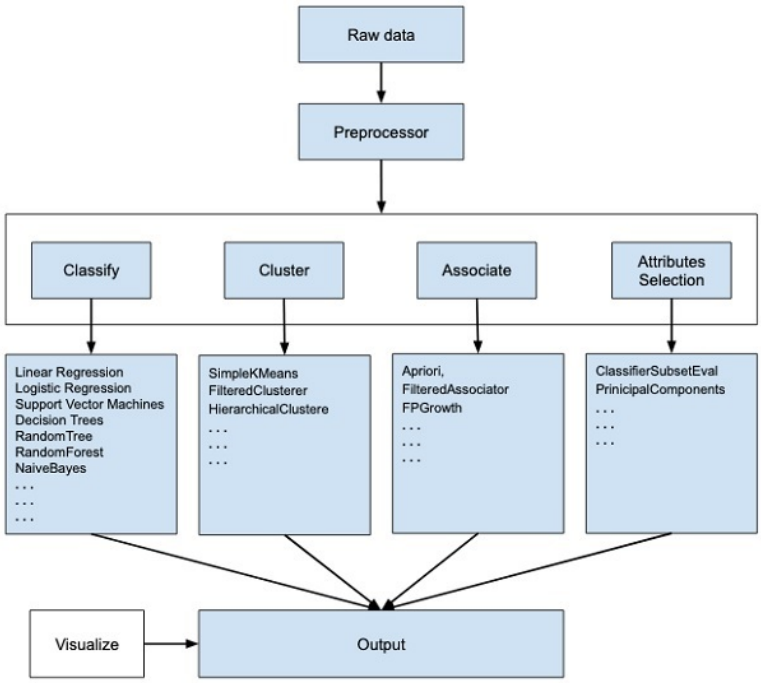


Figura 4.2: Resumen de las funcionalidades de Weka. Imagen recuperada de https://www.tutorialspoint.com/weka/what_is_weka.htm

visualización para inspeccionar los datos. Los distintos modelos pueden aplicarse al mismo conjunto de datos, permitiendo la comparación de los resultados.

Página web de la herramienta: https://waikato.github.io/weka-wiki/

PyCharm

PyCharm es uno de los IDEs[8] con soporte para Python más completos y exhaustivos, convirtiéndolo en uno de los más populares IDEs. Este éxito proviene en gran medida de que la empresa desarrolladora de este *software* es JetBrains, el desarrollador detrás del popular IDE Intellij IDEA, uno de los 3 IDEs más grandes de Java.

Disponible como una aplicación multiplataforma, PyCharm es compatible con los siguientes sistemas operativos: Windows, Linux y MacOS.

[8]Entorno de desarrollo integrado, sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz gráfica.

Proporciona soporte para las versiones de Python 2.x (descontinuada desde 2021) y 3.x. Provee de un elevado número de módulos, paquetes y herramientas diseñadas para optimizar el desarrollo del código, al mismo tiempo que reduce el esfuerzo necesario para ello. Siendo totalmente personalizable en función de los requisitos de desarrollo y las preferencias personales. Cuenta con:

- *Debugger* gráfico.
- Validación de pruebas unitarias.
- Soporte integrado para sistemas de control de versiones, VCS.
- Soporte para análisis de datos con Anaconda.

PyCharm permite trabajar con múltiples bases de datos directamente sin necesidad de utilizar terceras aplicaciones en forma de intermediarias. A pesar de que está diseñado para Python, tiene soporte para HTML, CSS, Javascript,...

Características y ventajas de PyCharm:

- Editor de código inteligente.
- Permite integrar nuevas herramientas.
- Soporte integrado para *Machine Learning* y *Data Science*.
- Soporte integrado para Google App Engine.
- Sistema de *debugging* y *testing*.
- Soporte para desarrollo de múltiples lenguajes.
- Navegación a través del código y del proyecto.
- *Refactoring*.
- Soporte para desarrollo remoto.
- Soporte de los principales *frameworks* de desarrollo web.
- Sistemas de versión de control.
- Generación de código específico de un lenguaje.

- Documentación directa de la API en uso.
- Plantillas para nuevos ficheros.
- Plantillas «vivas».
- Ayuda para la importación de bibliotecas.
- Corrección y optimización del código.

Entre las principales desventajas se encuentran:

- Requisitos mínimos elevados.
- Precio de la licencia de uso elevado.
- Curva de aprendizaje.

Página web de la herramienta: https://www.jetbrains.com/pycharm/

TEXMaker

TEXMaker es un *software* multiplataforma, disponible en Windows, Linux y MacOs; el cual permite la edición de documentos LATEX. Integra en él todas las herramientas necesarias para la creación, modificación y desarrollo de documentos escritos en LATEX. Entre sus características principales destacan el soporte a unicode, comprobación ortográfica «en directo», sugerencias y principalmente un visor del fichero pdf que se está editando. Además en caso de haber errores en la compilación, posee registros indicando el error y la línea que lo ha producido, facilitando el *debugging*.

Añadido a todas las ventajas que oferta, es completamente gratuito.

Página web de la herramienta: https://www.xm1math.net/texmaker/

FileZilla

FileZilla es un *software* multiplataforma, disponible en Windows, Linux y MacOs; totalmente gratuito, el cual incluye funcionalidades del protocolo FTP. Permite cargar y descargar archivos desde y hacia un servidor, siendo la opción líder en el mercado si se desean hacer múltiples transferencias manteniendo un elevado rendimiento. Ofrece soporte a FTP,

SFTP, proveedores *cloud* (Google Drive, OneDrive, Amazon S3 y DropBox). Como característica diferenciadora a otros programas similares, da soporte tanto a IPv4 como a IPv6.

Además, puede utilizarse como servidor FTP, siendo compatible con HTTP/1.1, SOCKS5, y FTP proxy.

Página web de la herramienta: https://filezilla-project.org

GitKraken

GitKraken es un potente *software* diseñado para realizar de forma gráfica todas aquellas tareas que serían realizadas mediante Git en la consola de comandos tradicional. Es una herramienta multiplataforma, con soporte para Windows, Linux y MacOS; permite de forma sencilla mantenerse al tanto de repositorios, *braches*, etiquetas, históricos, realizar *commits*, etcétera. Entre sus características más relevantes destacamos:

- Soporte para múltiples perfiles.
- Integración nativa con GitHub Enterprise, GitLab, Bitbucket y VCTS.
- Edición y visualización de ramas, *merging*, histórico de *commits*.
- Simplicidad en cuanto a las acciones *merge*, *rebase* y *push*. También admite crear, clonar y añadir repositorios de forma remota, así como ver y crear *pull requests*.
- Creación de repositorios, favoritos, organización de productos y grupos.
- Incluye editor *buit-in* de código.
- Muestra la diferencia entre los ficheros con cambios respecto a lo ya cometido.
- Soporte de Gitflow, Git Hooks y LFS.

Página web de la herramienta: https://www.gitkraken.com

ZenHub

ZenHub es una herramienta de gestión de proyectos ágiles integrados en GitHub. Acerca la gestión de proyectos al código, reduciendo los cambios de contexto y aumentando la productividad del equipo de desarrollo. Permitiendo la interconexión de múltiples repositorios en un solo *Workspace*, lo que otorga una visión global de todo el trabajo realizado y por realizar. Ayuda a un equipo de desarrollo a organizar, priorizar, asignar y realizar el seguimiento de las tareas de todo el equipo, adjuntando información estadística y reportes de cada *sprint*.

Se encuentra disponible o como extensión tanto para Google Chrome como para Mozilla Firefox, en sus correspondientes *marketplaces*, o desde la propia página web de ZenHub.

Página web de la herramienta: https://www.zenhub.com

Visual Paradigm

Visual Paradigm es una herramienta UML-CASE: Ingeniería de *Software* Asistida por Computación. Desarrollada para soportar el ciclo de vida completo del proceso de desarrollo *software* a través de la representación de todo tipo de diagramas.

Permitiendo el modelado de modelado de diagramas UML (entre otros), dando soporte principalmente a diagramas de clases, casos de uso,secuencia, estados, actividad, paquetes, etc.

Página web de la herramienta: https://www.visual-paradigm.com/

Aspectos relevantes del desarrollo del proyecto

En esta sección se van a detallar los aspectos más relevantes que han ido surgiendo a lo largo del desarrollo del proyecto. Este a sido un desarrollo *software* y como tal ha estado lleno de retos y decisiones técnicas que han tenido que ir tomándose. La formación ha sido un punto de inflexión, sumado a la aplicación de buenas prácticas, ha resultado en un producto de calidad.

5.1. Investigación

Uno de los componentes principales que posee el proyecto es el apartado de investigación en el campo del Aprendizaje Automático, concretando un poco más, el efecto de la aplicación de técnicas de selección de instancias en el aprendizaje semi-supervisado.

Uno de los principales retos a los que el autor se ha enfrentado en una primera instancia ha sido adquirir una rápida capacidad de lectura y asimilación de documentación científica, si bien el idioma (inglés) no ha supuesto una barrera, el no poseer experiencia en realizando estas tareas generó una necesidad que tuvo que ser solventada poco a poco. En los últimos meses del proyecto la agilidad ya estaba ahí, resultando en tareas más fluidas y veloces.

5.2. Metodología SCRUM

Como ya se mencionó en la Sección 4.1, el proyecto se ha realizado siguiendo una metodología ágil, permitiendo trabajar en *sprints* de manera que el trabajo de cada *sprint* se encuentre correctamente fundamentado al inicio de cada uno, permitiendo trabajar con una mayor eficiencia, priorizar el trabajo en función las tareas existentes, y disponer de versiones funcionales a la vez que se sigue con el desarrollo del proyecto.

Si bien se conocía SCRUM de manera puramente teórica, el haber trabajado siguiendo esta metodología ha permitido ganar experiencia en modelos de desarrollo ágiles, diferente de los clásicos a los que se acostumbraba, y tal y como se define, el trabajo se ha visto afectado de forma positiva con esta aproximación.

Se disponían de cerca de 9 meses (máximo) para cumplimentar el proyecto, un tiempo mucho más elevado del habitual para este tipo de proyectos, es por ello que se han invertido múltiples *sprints* para validación y calidad, permitiendo asegurar que los pasos dados son correctos y seguir construyendo sobre seguro.

Una de las principales dificultades encontradas en este campo es la asignación de puntos de historia, el definir en base a un nombre de tarea y una descripción el tiempo que va a ser necesario invertir para cumplir con ella no ha sido siempre muy preciso. El no poseer experiencia previa en proyectos de esta envergadura propició que se considerara la equivalencia de 1 punto de historia igual a 45 minutos o menos. Y si bien ha habido múltiples *sprints* en los cuales se ha conseguido seguir esta relación, no ha sido algo lineal en el proyecto, debido a que se tuvo en cuenta la experiencia que se iba consiguiendo a la hora de asignar los siguientes puntos de historia, y ha habido múltiples *sprints* desfasados por lo alto.

5.3. Actualización y modificación de un *software* pre-existente

Posiblemente será el aspecto relevante más cercano a el futuro laboral de cualquier alumno. Supuso todo un reto el coger un proyecto que había sido desarrollado por 5 personas y «hacerlo propio». Como es lógico cada persona programa de una forma diferente para alcanzar la misma

funcionalidad, y ya no solo eso, sino la propia temática del proyecto era desconocida, por lo tanto supuso un esfuerzo doble.

La no existencia de una documentación, ni técnica ni formal, acerca del propio *software* no ayudó a obtener una visión del proyecto más allá de la estructura de paquetes y nombres. Fue algo que se echó en falta, y que con la finalización de este proyecto, ya posee.

5.4. Desarrollo Web

A lo largo del Grado no se ha impartido docencia en cuanto a desarrollo web se refiere, siendo toda una desventaja a la hora de actualizar y modificar `UBUMLaaS`. Trabajar con una REST API era algo hasta enero desconocido, y la curva de aprendizaje no ha sido especialmente sencilla. Han sido necesarias numerosas horas para familiarizarse tanto con el proyecto como con la forma de trabajo que poseen las aplicaciones que siguen esta arquitectura.

A la hora de integrar nuevas modificaciones sobre el *backend* del proyecto no supuso grandes problemas, fue un trabajo laborioso como es de esperar, pero al ser Python ya se estaba acostumbrado al lenguaje de programación. La dificultad llegó con el *frontend* lenguajes de marcas como HTML o CSS, y de programación como JavaScript, no eran desconocidos, pero no se habían utilizado nunca en grandes proyectos; es por ello que hubo tareas que se vieron retrasadas.

Destacar que como ha sido la tónica general del proyecto, y así queda recogido en el resumen de numerosos *sprints* en su anexo correspondiente, a medida que el equipo de desarrollo se familiarizaba con el lenguaje y entorno de aplicación, la velocidad de implementación de funcionalidades crecía de forma exponencial.

La web ha sufrido una renovación completa, siendo necesario reescribir todos los documentos HTML, si bien se tuvo desde un inicio en cuenta que ciertas páginas tuvieran un diseño similar al que ya poseían anteriormente para seguir con el diseño inicial. El nuevo diseño soporta múltiples resoluciones de pantalla, habiendo sido priorizadas aquellas soportadas en tabletas sobre las de dispositivos móviles.

Funcionalidades «sobre la marcha»

En el momento de desarrollo de la parte de administración de UBUMLaaS surgían cada día nuevas funcionalidades que parecían correctas o adecuadas para implementar y añadir, permitiendo a los administradores tener más control del que iban a poseer una vez pase a producción la versión creada.

Muchas de estas nuevas funcionalidades se han llevado a cabo y se encuentran disponibles y completamente integradas con la plataforma, si bien muchas otras no, se han quedado como trabajos futuros. Aquellas que se han quedado sin implementar fueron clasificadas con una relevancia baja, siendo priorizadas aquellas que se consideraron más útiles o curiosas.

5.5. PIP

IS-SSL es una biblioteca desarrollada con la finalidad de que sea útil a la mayor cantidad de desarrolladores posible, siguiendo las directrices del *Open Source*, es por ello que se encuentra disponible a través del gestor de paquetes de Python, PIP.

El proyecto también puede ser descargado directamente desde el repositorio de GitHub y realizar todas las instalaciones de dependencias necesarias a través del fichero de *requeriments.txt* o creando un entorno de Conda (a través del fichero *is-ssl.yml*).

5.6. Docker

UBUMLaaS se encuentra desplegado en la Universidad de Burgos sobre un servidor directamente, pero con el fin de permitir que se despliegue en cualquier equipo y que la configuración específica sea mínima, se ha creado un contenedor Docker en el cuál reside la última versión de la aplicación.

El futuro de la aplicación podría ser únicamente en Docker perfectamente, siendo mucho más sencillo desplegarla en diferentes servidores, pero en la actualidad ese no es el enfoque deseado, por lo que se dejó para hacer al final en caso de que sobrara tiempo, como ha sucedido.

5.7. Validación de la integridad de los algoritmos implementados

Todos los algoritmos los cuáles se encuentran disponibles en `IS-SSL` han sido validados y refinados a lo largo de múltiples iteraciones de trabajo con el fin de garantizar su integridad, de forma que se puede asegurar que reportan resultados tal y como el *paper* original lo presentó.

- Los algoritmos de selección de instancias han sido validados contra los homónimos correspondientes proporcionados por `Weka`, *software* de ML desarrollado por la Universidad de Waikato.
- Los algoritmos de aprendizaje semi-supervisado han sido validados contra los implementados por el grupo de investigación ADMIRABLE de la Universidad de Burgos.

5.8. Experimentación de filtros de ruido para aprendizaje semi-supervisado

El aprendizaje semi-supervisado se basa en el entrenamiento de un modelo a partir de un pequeño número de prototipos etiquetados y uno mucho mayor de prototipos no etiquetados. Gracias a los innumerables campos de aplicación que posee, en los últimos años ha recibido mucha atención de la comunidad científica [72].

Dependiendo el dobjetivo final de los métodos, SSL puede ser aplicado en problemas de clasificación semi-supervisada [16] o *clustering* semi-supervisado [48]. La primera puede dividirse en dos aproximaciones ligeramente diferentes [18], el aprendizaje transductivo y el inductivo.

En el aprendizaje transductivo, el problema es abordado desde la predicción de las etiquetas de las instancias no etiquetadas, dadas de antemano, teniendo en cuenta tanto los datos no etiquetados como los que sí lo están para entrenar un clasificador. Mientras que en el aprendizaje inductivo se consideran los datos etiquetados y no etiquetados como datos de entrenamiento, siendo su objetivo el predecir los ejemplos no vistos.

La contrapartida se presenta en forma de que numerosos estudios empíricos demuestran como existen situaciones en las cuales el uso de esos

datos no etiquetados pueden degradar el modelo. Siendo aconsejable el poder explotar los datos de forma segura.

Es por ello que múltiples estudios [81, 30, 40] ya se han adentrado en el aprendizaje semi-supervisado seguro. Entendiendo por *seguro* que el rendimiento de la generalización nunca es estadísticamente peor que los métodos que sólo utilizan datos etiquetados [41].

La experimentación llevada a cabo posee como objetivo validar la hipótesis:
«¿Se obtiene una selección más segura en aprendizaje semi-supervisado gracias a la aplicación de métodos de selección de instancias?».

Introduciendo como novedades nuevos métodos de selección de instancias, así como clasificadores, hasta ahora no evaluados en problemas de esta naturaleza.

Experimentación

La experimentación ha sido realizada utilizando la aproximación propuesta en [39], bajo la cual el etiquetado de las instancias se produce en base a picos de densidad y la reducción del mismo mediante el uso de un filtrado de ruido previo. Así mismo, también se ha realizado la aproximación mediante el uso de *self-training*.

Para la realización de la experimentación se utilizan 18 conjuntos de datos seleccionados del repositorio de la Universidad de California Irvine (UCI), la descripción de los diferentes conjuntos de datos se encuentra en la Tabla 5.3. Los experimentos son realizados con diferentes porcentajes de número de instancias etiquetadas sobre el total, disponibles en la Tabla 5.4.

En todos los experimentos, se realiza una validación cruzada de 10 *folds*, con el fin de obtener los resultados de la experimentación se utilizan las métricas *accuracy* (ACC), *f1 score*, y el error cuadrático medio.

$$ACC = \frac{\#\ \text{tp}}{\#\ \text{Total de instancias}} \tag{5.1}$$

$$f1\text{-}score = \frac{\#\ \text{tp}}{\#\ \text{tp} + \frac{1}{2}\left(\#\ \text{fp} + \#\ \text{fn}\right)} \tag{5.2}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - Y_j\right)^2 \tag{5.3}$$

Métricas alcanzadas a partir de la matriz de confusión, esta es una herramienta que permite analizar los resultados de cómo trabaja un algoritmo de aprendizaje automático, repartiendo la información en columnas con las predicciones realizadas y en filas para el número real de instancias de cada clase. Donde TP representa los *true positive*, TN los *true negative*, FP los *false positives* y FN los *false negatives*.

El conjunto de datos es dividido en 10 *folds*, 9 de ellos para el conjunto de entrenamiento y 1 para el conjunto de prueba. Se utiliza selección estratificada aleatoria para la separación de los datos en entrenamiento y validación.

Anteriores artículos [39] proponen esta misma aproximación pero utilizando únicamente el filtro propio *ENaNE*, y el clasificador base *k-Nearest Neighbors*, tal y como se muestra en la Tabla 5.1 esta aproximación utiliza 3 clasificadores base con el fin de obtener una mayor diversidad de soluciones. Además, se utilizan los métodos de selección de instancias presentados en la Tabla 5.2. Con todo esto se obtiene un espectro mucho más amplio de soluciones comparables, permitiendo validar la hipótesis planteada.

El procedimiento seguido es directo y sencillo, mediante el uso de una semilla aleatoria, se asegura que todos los clasificadores y filtros poseen los mismos datos en cada *n-ésima* iteración. El pseudocódigo se encuentra dispoible en el Algoritmo 12.

Algoritmo 12: Pseudocódigo del proceso de experimentación.

```
random_seed ← XX
for dataset en datasets do
    for clasificador en clasificadores do
        for filtro en filtros do
            for fold en CV do
                Instanciar el modelo con sus parámetros.
                Entrenar el modelo.
                Predecir los datos de test.
                Cálculo de métricas.
                Guardar las métricas.
            end for
        end for
    end for
end for
```

Clasificador base	**Parámetros**
k Nearest Neighbors	$k = 3$
Decission Tree	*criterion* = gini
Gaussian NB	

Tabla 5.1: Clasificadores base utilizados en la experimentación.

Filtro	**Parámetros**	**Referencia**
ENN	*k-NN* = 3, *distance function* = *euclidean*	[77]
LSSm		[37]
ENaNE		[39]

Tabla 5.2: Filtros utilizados en la experimentación.

Conjuntos de datos	Instancias	Atributos	Clases	Abreviación
User Knowledge Modeling	403	5	5	UKM
Thoracic Surgery Data	470	16	2	TSD
SPECTF Heart	267	44	2	SPH
Sonar	208	60	2	SON
Image Segmentation	2.310	19	7	IMS
Pendigits	3.498	16	10	PEN
Liver	345	6	2	LIV
Ionosphere	351	34	2	ION
Glass	214	10	7	GLA
Ecoil	336	7	8	ECO
Blood Transfusion Service Center	748	4	2	BTSC
Dermatology	366	34	6	DER
Satimage	6.435	36	7	SAT
Wireless Indoor Localization	2.000	7	4	WIL
Wholesale Customers	440	7	2	WHC
Indian Liver Patient Data set	583	10	2	ILPD
Breast Tissue	106	9	6	BRT
Letter	15.534	16	26	LET

Tabla 5.3: Descripción de los conjuntos de datos usados en la experimentación.

Conjunto de datos	5 %	10 %	15 %	20 %	25 %	30 %	35 %	100 %
UKM	20	40	60	80	100	120	141	403
TSD	23	47	70	94	117	141	164	470
SPH	13	26	40	53	66	80	93	267
SON	10	20	31	41	52	62	72	208
IMS	115	231	346	462	577	693	808	2.310
PEN	174	349	524	699	874	1.049	1.224	3.498
LIV	17	34	51	69	86	103	120	345
ION	17	35	52	70	87	105	122	351
GLA	10	21	32	42	53	64	74	214
ECO	16	33	50	67	84	100	117	336
BTSC	37	74	112	149	187	224	261	748
DER	18	36	54	73	91	109	128	366
SAT	321	643	965	1.287	1.608	1.930	2.252	6.435
WIL	100	200	300	400	500	600	700	2.000
WHC	22	44	66	88	110	132	154	440
ILPD	29	58	87	116	145	174	204	583
BRT	5	10	15	21	26	31	37	106
LET	776	1.553	2.330	3.106	3.883	4.660	5.436	15.534

Tabla 5.4: Relación de el número de instancias de cada conjunto de datos con los diferentes porcentajes de etiquetado utilizados.

Resultados

El análisis de los resultados a los que podemos llegar se realiza en función de cada estimador base, ya que se han seleccionado 3 de naturaleza distinta, no se pueden comparar de manera tan simple entre ellos.

La aproximación seguida para conocer qué filtro (o ninguno) reporta mejores resultados para cada conjunto estimador base, han sido los *rankings* medios para cada conjunto de datos y cada porcentaje de etiquetado. En la Figura 5.1 se muestra un resumen de todos los filtros para cada estimador[9].

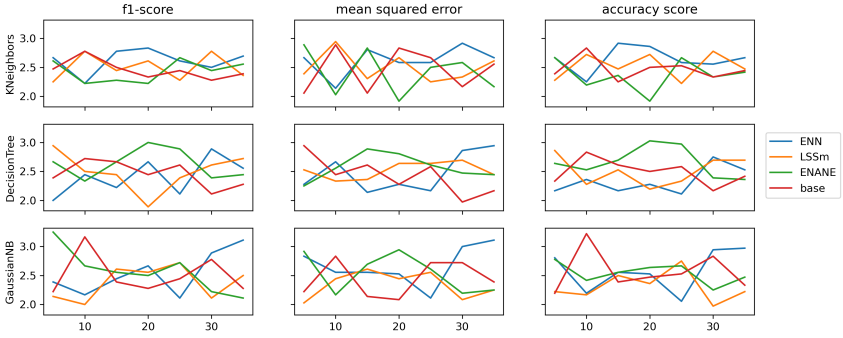


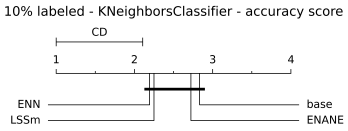
Figura 5.1: Resumen en función del clasificador y filtro.

En la Figura 5.1 no se aprecia a «simple vista» tendencia alguna para ningún clasificador base y método de selección de instancias, que destaque sobre el resto. Si bien se puede intuir que `LSSm` tiene a mejorar según se poseen más instancias etiquetadas, no es una conclusión válida, es por ello que se realizan tests estadísticos para comprobar si existen diferencias significativas entre los mismos.

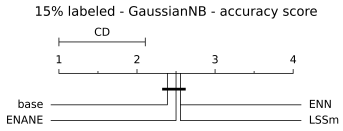
El cálculo de los *rankings* medios se ha validado gracias al uso de la herramienta `KEEL`[10] [6, 5]. Con la verificación de que los cálculos son correctos, se procede a realizar el *test* de Nemenyi [44] como *test post-hoc*, con el objetivo de encontrar las diferencias significativas entre clasificadores

[9] *base* hace referencia a el resultado reportado por el estimador sin la aplicación de ningún método de selección de instancias sobre el conjunto de datos etiquetados.

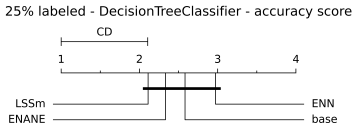
[10] *Knowledge Extraction based on Evolutionary Learning*. Herramienta de software Java que puede utilizarse para un gran número de tareas diferentes de descubrimiento de datos de conocimiento. KEEL proporciona una sencilla interfaz gráfica de usuario basada en el flujo de datos para diseñar experimentos con diferentes conjuntos de datos y algoritmos de IA.



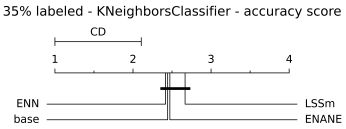
(a) Diagrama *Critical Difference* para *k-Nearest Neighbors*.



(b) Diagrama *Critical Difference* para *Gaussian NB*.



(c) Diagrama *Critical Difference* para *Decission Tree*.



(d) Diagrama *Critical Difference* para *k-Nearest Neighbors*.

Figura 5.2: The average and standard deviation of critical parameters: Region R4

y los filtros utilizados. Este último paso viene propiciado como resultado de que el *test* de Friedman [25] rechazara la hipótesis nula.

Por motivos de espacio se muestra un resumen de las figuras obtenidas: 5.2a, 5.2b y 5.2c, cada par clasificador-porcentaje de instancias etiquetadas dispone de su propio diagrama de *Critial Difference* disponibles desde el siguiente enlace. De igual manera los `CSVs` resultantes del proceso de experimentación, los *logs*, los *rankings* medios, ..., se encuentran disponibles en el correspondiente directorio del repositorio de `IS-SSL`.

Conclusiones

Tal y como se apreciaba ya en la Figura 5.1, no existen diferencias significativas entre los diferentes métodos de selección de instancias en relación con el aprendizaje semi-supervisado seguro. Los diagramas representando la Diferencia Crítica 5.2a, 5.2b, 5.2c, y todos los demás no añadidos, no reportan en ningún caso un claro vencedor sobre los demás, para ningún clasificador base analizado.

Estos resultados nos permiten llegar a la conclusión de que si bien los tiempos de ejecución son ligeramente inferiores en función del método de selección de instancias utilizado (en función de la «agresividad» con la que elimine instancias), no existe un ganador.

Trabajos relacionados

En esta Sección se van a comentar los trabajos relacionados desde dos ópticas, la primera de ellas desde el *Machine Learning As A Service*, MLaaS; y la segunda desde el punto de vista del Aprendizaje Semi-Supervisado Seguro (*Safe-SSL*).

6.1. *Machine Learning As A Service*

Frameworks y bibiotecas

Aunque está categorizado con dos términos, se puede entender un *framework* de *Machine Learning* como una herramienta, biblioteca o interfaz que proporciona a los desarrolladores facilidades para crear modelos de aprendizaje automático.

1. ***Scikit-Learn***. Coloquialmente conocido como «Sklearn», es la biblioteca más útil y robusta para aprendizaje automático implementada en Python. Se trata de un *software open source*. Entre la multitud de algoritmos que provee, destacan varios para problemas de clasificación, regresión y *clustering*; incluyéndose además máquinas de soporte vectorial (SVM), *random forests*, *gradient boosting*, *k-means* y DBSCAN.

   Si bien proporciona soporte para el aprendizaje semi-supervisado, este es mínimo, pudiendo encontrar únicamente *Self-Trainig* entre los clasificadores disponibles.

A pesar de estar escrito en su mayor parte en Python, y el uso que le da a la biblioteca de NumPy[11] para el cálculo de operaciones algebraicas en entornos de alto rendimiento, el *core* de la biblioteca está escrito en Cython[12] para mejorar el rendimiento.

2. ***Tensor Flow.*** Biblioteca desarrollada por Google, sigue una filosofía *open source*. Tensor Flow es utilizada para el cálculo numérico utilizando gráficos de flujo. En las gráficas los nodos representan operaciones matemáticas mientras que los bordes representan las matrices de datos multidimensionales.

   Mediante su arquitectura flexible permite la implementación del cálculo en una o varias CPU o GPU, en servidores, o equipos móviles, todo con una sola API[13]. Su diseño esta principalmente enfocado para la resolución de problemas mediante redes neuronales, pero es lo suficientemente general como para ser aplicable a una amplia variedad de dominios.

3. ***Torch.*** Marco de cálculo científico con amplio soporte para algoritmos de aprendizaje automático el cual da prioridad al uso de GPU sobre CPU, se trata de un *software open source*. Una de sus características principales es el rendimiento y eficiencia que proporciona, esto se debe a que está escrito en LuaJIT y una implementación subyacente de C/CUDA.

4. ***Theano.*** biblioteca de Python la cual da soporte a la definición de expresiones matemáticas empleadas en aprendizaje automático, la optimzación de estas expresiones y la evaluación de la eficiencia con el uso de GPU. Siendo capaz de rivalizar implementaciones escritas en C. Theano está distribuida bajo licencia BSD[14].

5. ***Veles.*** Escrito en C++, sus aplicaciones se encuentran dentro del marco del aprendizaje profundo. A pesar de ello utiliza Python para la

[11] biblioteca de Python que da soporte al uso de vectores y matrices con una gran dimensionalidad. Además proporciona una gran colección de funciones matemáticas de alto nivel para operar con ellas.

[12] Compilador estático optimizado tanto para Python como para el lenguaje Cython extendido, (basado en Pyrex). Permitiendo escribir extensiones de C para Python de manera tan sencilla que casi parece Python.

[13] Conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el *software* de las aplicaciones, permitiendo la comunicación entre dos aplicaciones a través deun conjunto de reglas.

[14] Licencia de *software* libre permisiva, de igual manera que la licencia MIT, posee menos restricciones que GPL, siendo muy cercana al dominio público.

automatización y coordinación de sus nodos. Sus objetivos principales son la flexibilidad y el rendimiento. Permitiendo la normalización de los datos antes de introducirlos en un cluster.

Veles permite entrenar redes convolucionales, redes recurrentes, redes totalmente conectadas y otras topologías populares.

6. ***H2O.*** Marco de aprendizaje automático *open source*. Orientado principalmente a negocios, implementa análisis predictivo para ayudar a la toma de decisiones basadas en datos y conocimientos. Proporciona herramientas únicas como el soporte agnóstico de bases de datos, una interfaz WebUI.

   H2O provee de modelos y soporte para Python, R, Java, JSON, Scala, JavaScript. Su *core* está escrito en Java.

Proveedores de MLaaS

El Aprendizaje Automático como servicio (MLaaS por sus siglas en inglés), es una tecnología de aprendizaje automático que es habitualmente adquirida de un tercero. Su funcionamiento es similar a SaaS (*Software as a Service*) o PaaS (*Platform as a Service*), *i.e.* un usuario utiliza los servicios de un tercero en lugar de los suyos propios.

1. ***Amazon Machine Learning.*** Servicio ofrecido por Amazon el cual proporciona todas las herramientas necesarias para utilizar modelos de aprendizaje automático sin necesidad de conocer todos los detalles y configuraciones de estos. No quedándose ahí, proporciona herramientas de análisis de datos y modelos pre-entrenados para casos de uso habituales como detección de fraude en aplicaciones móviles.

   Encaja a la perfección con proyectos o necesidades que necesitan interacción en tiempo real. A pesar de que únicamente proporciona soporte a problemas de clasificación binaria o multi-etiqueta, y regresión; una de sus funcionalidades más «potentes» viene dada por la capacidad del propio servicio de seleccionar el algoritmo que mejor se adecua al problema dado.

2. ***SageMaker.*** Entorno de aprendizaje automático con el objetivo de simplificar el trabajo a los científicos de datos, para ello proporciona herramientas que permiten la creación y despliegue de forma rápida de modelos.

Proporciona multitud de modelos que su predecesor (Amazon *Machine Learning*) no poseía, como es el caso del aprendizaje no supervisado. Siendo la evolución natural para aquellas empresas que ya utilizan los servicios web de Amazon (AWS).

3. ***Microsoft Azure AI Platform.*** Proporciona una plataforma unificada con todas las API correspondientes a técnicas de aprendizaje automático y servicios de infraestructura en Azure.

   Destaca *Azure Machine Learning* como entrono por excelencia para el manejo de conjuntos de datos, creación, entrenamiento y despliegue de modelos. Permitiendo que desde la interfaz web y con muy poca codificación necesaria, se puedan crear modelos. Ofreciendo soporte a cerca de cien métodos para problemas de clasificación binaria y multi-etiqueta, detección de anomalías, regresión, recomendación, análisis de textos, y como único algoritmo de *clustering*, *k-means*.

4. ***Google Cloud ML.*** Plataforma de *Machine Learning* basada en la nube que sugiere un enfoque sin código para construir soluciones basadas en datos. Fue diseñado para que tanto los recién llegados como los ingenieros fueran capaces de construir modelos personalizados. Como es habitual, ofrece también un conjunto de modelos pre-construidos, a través de un conjunto de API.

5. ***IBM Watson Machine Learning Studio.*** Aporta una interfaz de procesamiento de datos y creación de modelos totalmente automatizada que apenas necesita formación para empezar a procesar los datos, preparar los modelos y desplegarlos en producción.

   La parte automatizada es capaz de resolver problemas de clasificación binaria y multi-etiqueta, y regresión. Permitiendo la opción de que el usuario elija el modelo de *Machine Learning* deseado o que sea el propio sistema el que infiera el que considera mejor a partir de los datos.

6. ***UBUMLaaS***. La Universidad de Burgos también dispone de su propio servicio de *MLaaS*, ver 4.2.

Comparativa entre MLaaS y UBUMLaaS

Tal y como se puede apreciar en la tabla 6.1, existen numerosos servicios soportados por los principales proveedores de *Machine Learning as a Service*, *e.g.* todos ellos permiten el etiquetado de datos con técnicas de clasificación y regresión.

Todas las herramientas anteriormente descritas han sido comparadas contra *UBUMLaaS*.

Servicios / Soporte	Amazon ML SageMaker	Microsoft Azure	Google AI Platform	IBM Watson	UBUMLaaS
Aprendizaje Supervisado	X	X	X	X	X
Aprendizaje Semi-Supervisado	X	X	X		X
Aprendizaje No Supervisado	X	X	X		
Clasificación	X	X	X	X	X
Regresión	X	X	X	X	X
Clustering	X	X		X	X
Recomendaciones	X	X	X		
Etiquetado de datos	X	X	X	X	X
Algoritmos pre-implementados	X	X	X		X
Coste (USD/h)	0,408	0,099	1,375	0,46	0

Tabla 6.1: Comparativa general entre proveedores de MLaaS.

6.2. Aprendizaje Semi-Supervisado Seguro

A pesar de que se han hecho multitud de aproximaciones y estudios sobre Clasificación Semi-Supervisada [23], los prototipos son habitualmente clasificados dependiendo de diferentes suposiciones relacionadas con la distribución de los ejemplos etiquetados y no etiquetados. Habitualmente los modelos se basan en la suposición de existencia de matrices y/o *clusters*. Si los datos corresponden a un *manifold*[15] de menor dimensionalidad que el espacio de entrada, es adecuado para suposición de *manifold* [73].

Siguiendo con esta idea en mente, la construcción de grafos permite determinar el comportamiento de los modelos, ya que dos prototipos conectados por una arista fuerte probablemente indique que ambos prototipos poseen la misma etiqueta [74]. La suposición de *cluster* supone que prototipos «similares» deberían tener las mismas etiquetas.

La aplicación de técnicas de «autoetiquetado» son aquellas que aprovechan un clasificador supervisado para etiquetar la clase desconocida y no hacen suposiciones específicas acerca de los datos de entrada [68]. Para ello lo habitual es entrenar un clasificador o un conjunto de ellos y posteriormente aprovechar el conocimiento adquirido por este(os) clasificador(es) para entrenar uno nuevo que produzca mejores resultados [10, 84].

Todos los modelos con los que habitualmente se trabaja se basan únicamente en el uso de aquellas instancias que se encuentran etiquetadas para obtener una mayor diversidad en los clasificadores, sin pararse a utilizar la gran y abundante información que se encuentra dentro de los prototipos no etiquetados [81]. Pero es aquí donde surge el problema real, no se tiene en cuenta que estos clasificadores iterativos también introducen ruido en el conjunto de datos etiquetados, es decir, clasifican mejor o peor pero no son seguros; todo ello propicia que en determinadas ocasiones el rendimiento empeore.

En [69] se propone el análisis de características de una gran variedad de filtros de ruido de diferente naturaleza, con el objetivo de mejorar el auto-entrenamiento en aprendizaje semi-supervisado orientado a problemas

[15] Término técnico que se utiliza para clasificar espacios de dimensión arbitraria. Para cada número entero existe un espacio plano llamado espacio euclidiano que tiene características muy similares al plano cartesiano. Esencialmente una generalización del espacio euclidiano tal que localmente (áreas pequeñas) es aproximadamente lo mismo que el espacio euclidiano pero el espacio entero no tiene las mismas propiedades del espacio euclidiano cuando se observa en su totalidad.

de clasificación. Muchos de los filtros propuestos ya se habían estudiado previamente en aprendizaje supervisado, pero el proceso de filtrado puede ser más difícil de realizar cuando se trata de problemas de aprendizaje semi-supervisado debido al reducido número de instancias que se poseen etiquetadas.

[69] comprueba como los filtros «globales», algoritmos CF e IPF, destacan como la familia de filtros con mejor rendimiento, mostrando que la concordancia de hipótesis de varios clasificadores también es robusta cuando se reduce la proporción de datos etiquetados disponibles. La mayoría de los enfoques locales necesitan más datos etiquetadas para rendir mejor. El uso de estos filtros ha dado lugar a un mejor rendimiento que el logrado por métodos de auto-formación como son SETRED y SNNRCE. Obteniendo como conclusión que el uso de filtros «globales» es muy recomendable en el campo en el que se enmarca tanto este como el citado trabajo.

Con el fin de trabajar con aprendizaje semi-supervisado seguro, en [81] se propone una nueva forma de trabajar con clasificadores supervisados en un *ensemble*, los cuales a partir de múltiples iteraciones y pasadas sobre el conjunto de datos etiquetados lo acabarán etiquetando de forma segura. Para ello los clasificadores son entrenados con conjuntos de datos extraídos de los prototipos etiquetados y los cuales han sido seleccionados entre aquellos que poseen una baja ambigüedad. Posteriormente se etiquetas aquellos prototipos para los cuales los clasificadores acuerdan mediante mayoría de la clase a la que corresponde y se reentrenan los modelos.

De la misma forma que se acaba de ver cómo hay trabajos en la literatura acerca de mejorar los métodos ya existentes de clasificación semi-supervisada, también existen métodos basados en *clusters* los cuáles eran brevemente introducidos al principio de esta sección. Uno de los mayores problemas que se encontraban éstos métodos era el poder generalizar para cualquier conjunto de datos independientemente de cuál fuese su distribución [4, 26].

Gracias al trabajo de [58] en el cual propone que para todo prototipo del conjunto de datos global (etiquetado y no etiquetado) el algoritmo calcula dos valores, su densidad local y la distancia a los puntos de mayor densidad local. Permitiendo descubrir la estructura real del espacio de datos, sin importar si la distribución de los datos es esférica o no, puede ser descubierta haciendo que cada prototipo apunte a su prototipo más cercano con una densidad local más alta.

Con base esta última aproximación, en [79] se propone una aproximación que integra la estructura descubierta basada en picos de densidad junto con el proceso de entrenamiento semi-supervisado, mediante el entrenamiento iterativo de un clasificador supervisado. Obteniendo las ventajas de:

- No se encuentra limitado por la distribución inicial de los datos etiquetados y del conjunto de datos general.
- Es un modelo constrictivo sin condiciones anteriores.
- Es un modelo adecuado para mejorar el rendimiento de cualquier algoritmo supervisado mediante el uso de grandes cantidades de datos.

Conclusiones y Líneas de trabajo futuras

En esta última sección se exponen las conclusiones finales recuperadas del proyecto realizado. Además, se añaden las líneas futuras que se pueden seguir para continuar con el desarrollo de las bibliotecas, y/o de `UBUMLaaS`.

7.1. Conclusiones

Conclusiones a las que se llega posterior al desarrollo del proyecto.

- Los objetivos del proyecto han sido cumplidos satisfactoriamente.
  - `IS-SSL`. Ahora la comunidad cuenta con dos bibliotecas las cuales proporcionan aquellos algoritmos más comúnmente utilizados en la literatura. Estas son públicas permitiendo que cualquier persona pueda utilizarlas.
  - `UBUMLaaS`. La aplicación cuenta con soporte para algoritmos de aprendizaje semi-supervisado. Además dispone de una «parte» de administración la cuál va a permitir a los usuarios con el rol de administrador ser capaces de no solo conocer el estado del sistema, sino también administrar usuarios y poseer estadísticas en tiempo real, todo ello sin necesidad de acceder a la base de datos «a mano».
    Tras la finalización del proyecto, `UBUMLaaS` cuenta con una documentación técnica de programador, lo cual favorecerá su evolución

y desarrollo futuro, así como la facilidad de mantenimiento y reducción de deuda técnica.

- La lectura de artículos científicos ha sido algo totalmente nuevo, siendo necesarias numerosas horas para su interpretación (sobretodo en los primeros), se debe destacar que según se avanzaba en el desarrollo del proyecto, la familiarización con éstos ha sido satisfactoria, permitiendo asimilar e implementar los algoritmos deseados con mayor facilidad.
- El haber desarrollado los algoritmos de `IS-SSL` en Python posee las ventajas de poseer multitud de bibliotecas orientadas a operaciones vectoriales y con *DataFrames*, su portabilidad, posee una baja curva de aprendizaje, y que se trata de un lenguaje de alto nivel.
- Las bibliotecas poseen una estructura que permite su escalabilidad, permitiendo que no sea un trabajo cerrado sino que se puedan ampliar y seguir evolucionando.
- El desarrollo de `UBUMLaaS` ha permitido ampliar el número de tecnologías conocidas, en un inicio se habían utilizado muy poco la mayor parte de las tecnologías utilizadas para el desarrollo del *frontend*. Pero a la hora de finalizar el trabajo ya se está familiarizado con ello y se posee una confianza y capacidades de desarrollar garantizando una calidad dada.
- A lo largo del desarrollo del proyecto se han utilizado diferentes herramientas *cloud*, éstas han permitido aumentar la calidad del producto final que se iba obteniendo en sucesivas iteraciones. Se deberían de haber definido en un primer momento antes de comenzar a desarrollar el proyecto, pero algunas de ellas eran desconocidas. Cabe destacar que el esfuerzo extra que ha tenido que ser invertido, en futuros proyectos resultará útil.

Como **conclusión final**, destacar la oportunidad de *aprendizaje* que el desarrollo de este proyecto ha supuesto. Desde el minuto uno se ha requerido dominar ciertas tecnologías vistas (y nuevas) a lo largo de todo el Grado, incluso a la hora de escribir la documentación. El número de horas que el proyecto iba a requerir se conocía desde un primer momento pero el razonamiento inicial de *«quedan 8 meses por delante, hay tiempo para todo»*, si bien se ha cumplido (el proyecto se ha terminado antes de las fechas de entrega), la disciplina de trabajo como si fuera un entorno laboral ha sido necesaria.

7.2. Líneas de trabajo futuras

`UBUMLaaS`, dada la morfología y arquitectura de la aplicación, existen numerosas opciones de mejora o líneas de trabajo futuras.

- Migración de la aplicación a `Kubernetes`, permitiendo el despliegue sobre *clusters*. Dada la naturaleza de la aplicación, es una evolución lógica.
- Aumentar el número de algoritmos soportados, incluyendo su implementación en diferentes lenguajes (actualmente soportados `IS-SSL`, `Scikit-Learn` y `Weka`).
- Aumentar el conjunto de pruebas de forma que abarquen tanto *backend* como *frontend*.
- Realizar un proceso de refactorización (únicamente en caso de que el proyecto vaya aumentar sus funcionalidades).
- Mejoras de *backend*:
  - Soporte de parseo de ficheros mediante URL.
  - Soporte de parseo de ficheros personalizados (separadores, saltos de línea, etc.).
  - Detección de entrenamientos idénticos repetidos con el fin de evitar repetir procesos demasiado largos en CPU.
  - Migración de la ejecución de los entrenamientos a ejecución en GPUs.
  - Proporcionar soporte de ejecución paralela sobretodo en aquellos procesos con validación cruzada.
- Mejoras de *frontend*:
  - Añadir un modo oscuro a la aplicación.
  - Soportar la descarga de históricos y estadísticas, tanto del estado del sistema como de los experimentos en ejecución.
  - Separar el perfil del usuario con sus estadísticas personales, de los conjuntos de datos y experimentos asociados al mismo.
  - Permitir el auto-refresco de la vista cuando un usuario se quede «esperando» a que el experimento finalice.

`IS-SSL`, el propio diseño de ambas bibliotecas sugiera una posible unión en el futuro, en caso de que se acaben comenzando a utilizar de forma conjunta y facilite su uso.

Para facilitar la aportación de la comunidad a las bibliotecas, se van a definir las convenciones, buenas prácticas, ... de forma que la evolución

de las bibliotecas mantenga una estructura y un código limpio y sobretodo, *fácil* de mantener. En esta misma línea se va a aplicar el método plantilla [64] con el fin de re-estructurar las bibliotecas, permitiendo crear un `core` común y desacoplar ciertas funcionalidades.

Una de las mejoras que se plantean para realizar a corto/medio plazo es la migración de los algoritmos a `Cython`, de manera que haya un aumento considerable del rendimiento. Otra opción que se propone es la modificación de los algoritmos para, en aquellas partes soportadas, corran en paralelo tanto mediante hilos, como mediante procesadores lógicos o reales.

Investigación. La investigación, como es lógico, no está ni cerca de estar terminada. El campo es muy amplio y quedan muchas preguntas por responder. Una de las principales mejoras que se puede realizar es hacer uso de *Random Forests* en lugar de árboles de decisión, evitando que queden hojas con una única instancia y afecten a la clasificación en el aprendizaje semi-supervisado [65].

Bibliografía

[1] Admirable blog. http://admirable-ubu.es.

[2] Pep 8 in python | what is the purpose of pep 8 in python?

[3] Scrum (desarrollo de software), 2022. https://es.wikipedia.org/wiki/Scrum-(desarrollo-de-software).

[4] Mathias M Adankon and Mohamed Cheriet. Help-training for semi-supervised support vector machines. *Pattern Recognition*, 44(9):2220–2230, 2011.

[5] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, Salvador García, Luciano Sánchez, and Francisco Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, 17, 2011.

[6] Jesús Alcalá-Fdez, Luciano Sánchez, Salvador Garcia, Maria Jose del Jesus, Sebastian Ventura, Josep Maria Garrell, José Otero, Cristóbal Romero, Jaume Bacardit, Victor M Rivas, et al. Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[7] Ricardo Barandela, Francesc J Ferri, and J Salvador Sánchez. Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(06):787–806, 2005.

[8] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1:2012, 2012.

[9] Jeevan Biswas. What is machine learning as a service (mlaas)?, 2018. https://analyticsindiamag.com/what-is-machine-learning-as-a-service-mlaas/.

[10] 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.

[11] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larranaga. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(5):216–233, 2015.

[12] Christos Boutsidis, Anastasios Zouzias, Michael W Mahoney, and Petros Drineas. Randomized dimensionality reduction for k-means clustering. *IEEE Transactions on Information Theory*, 61(2):1045–1062, 2014.

[13] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172, 2002.

[14] Pierre Carbonnelle. Pypl popularity of programming language.

[15] H Frank Cervone. Understanding agile project management methods using scrum. *OCLC Systems & Services: International digital library perspectives*, 2011.

[16] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[17] Peter Chapman, Janet Clinton, Randy Kerber, Tom Khabaza, Thomas P. Reinartz, Colin Shearer, and Richard Wirth. Crisp-dm 1.0: Step-by-step data mining guide. 2000.

[18] Ke Chen and Shihai Wang. Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):129–143, 2010.

[19] Potomac Two Crows Corporation. *Introduction to data mining and knowledge discovery.* Two Crows, 1999.

[20] Sanjoy Dasgupta, Michael L Littman, and David McAllester. Pac generalization bounds for co-training. *Advances in neural information processing systems*, 1:375–382, 2002.

[21] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[22] IBM Cloud Education. What is data mining?, 2021.

[23] Jesper Engelen and Holger Hoos. A survey on semi-supervised learning. *Machine Learning*, 109, 02 2020.

[24] Python Software Foundation. History and license.

[25] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[26] Haitao Gan, Nong Sang, Rui Huang, Xiaojun Tong, and Zhiping Dan. Using clustering analysis to improve semi-supervised classification. *Neurocomputing*, 101:290–298, 2013.

[27] Geoffrey Gates. The reduced nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 18(3):431–433, 1972.

[28] Zoubin Ghahramani and Michael I Jordan. Supervised learning from incomplete data via an em approach. In *Advances in neural information processing systems*, pages 120–127, 1994.

[29] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *OTM Confederated International Conferences On the Move to Meaningful Internet Systems*, pages 986–996. Springer, 2003.

[30] Lan-Zhe Guo, Zhen-Yu Zhang, Yuan Jiang, Yu-Feng Li, and Zhi-Hua Zhou. Safe deep semi-supervised learning for unseen-class unlabeled data. In *International Conference on Machine Learning*, pages 3897–3906. PMLR, 2020.

[31] David J Hand. Principles of data mining. *Drug safety*, 30(7):621–622, 2007.

[32] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

[33] Mathworks Inc. Supervised learning.

[34] JavaTPoint. Introduction to semi-supervised learning - javatpoint.

[35] Vijay Kotu and Bala Deshpande. Chapter 2 - data mining process. In Vijay Kotu and Bala Deshpande, editors, *Predictive Analytics and Data Mining*, pages 17–36. Morgan Kaufmann, Boston, 2015.

[36] Erik G Learned-Miller. Introduction to supervised learning. *I: Department of Computer Science, University of Massachusetts*, 2014.

[37] Enrique Leyva, Antonio González, and Raúl Pérez. Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. *Pattern Recognition*, 48(4):1523–1537, 2015.

[38] Cen Li and Gautam Biswas. Unsupervised learning with mixed numeric and nominal data. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):673–690, 2002.

[39] Junnan Li, Qingsheng Zhu, and Quanwang Wu. A self-training method based on density peaks and an extended parameter-free local noise filter for k nearest neighbor. *Knowledge-Based Systems*, 184:104895, 2019.

[40] Yu-Feng Li, James T Kwok, and Zhi-Hua Zhou. Towards safe semi-supervised learning for multivariate performance measures. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[41] Yu-Feng Li and De-Ming Liang. Safe semi-supervised learning: a brief introduction. *Frontiers of Computer Science*, 13(4):669–676, 2019.

[42] Enrico Lucon and Enrico Lucon. *New Software for the Statistical Analysis and Qualification of NIST Charpy Verification Specimen Lots.* US Department of Commerce, National Institute of Standards and Technology, 2018.

[43] Onesmus Mbaabu. Clustering in unsupervised machine learning.

[44] Peter Bjorn Nemenyi. *Distribution-free multiple comparisons.* Princeton University, 1963.

[45] National Institute of Standars and Technology. *Engineering Statistics*, chapter Product and Process Comparisons. U.S. Department of Commerce, 2012.

[46] J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143, 2010.

[47] Alfonso Palmer, Rafael Jiménez, and Elena Gervilla. Data mining: Machine learning and statistical techniques. *Knowledge-Oriented Applications in Data Mining, Prof. Kimito Funatsu (Ed.)*, pages 373–396, 2011.

[48] Witold Pedrycz. Algorithms of fuzzy clustering with partial supervision. *Pattern recognition letters*, 3(1):13–20, 1985.

[49] Dan Radigan. El backlog del producto: la lista de tareas pendientes definitiva, 2021.

[50] Daniel Puente Ramírez. Contenedor docker con ubumlaas desplegado., 2022. https://is.gd/cVe0DP.

[51] Daniel Puente Ramírez. Instance selection pypi, 2022. https://pypi.org/project/InstanceSelectionDNX/.

[52] Daniel Puente Ramírez. Semi-supervised learnig and instance selection, 2022. https://github.com/dpr1005/Semisupervised-learning-and-instance-selection-methods.

[53] Daniel Puente Ramírez. Semi-supervised learning pypi, 2022. https://pypi.org/project/SemiSupervisedLearningDNX/.

[54] Daniel Puente Ramírez. Ubumlaas, 2022. https://github.com/dpr1005/UBUMLaaS.

[55] Joel Ratsaby and Santosh S Venkatesh. Learning from a mixture of labeled and unlabeled examples with parametric side information. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 412–417, 1995.

[56] G Ritter, H Woodruff, S Lowry, and T Isenhour. An algorithm for a selective nearest neighbor decision rule (corresp.). *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

[57] Julio Roche. Scrum: roles y responsabilidades, 2020.

[58] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *science*, 344(6191):1492–1496, 2014.

[59] Guido van Rossum, Barry Warsaw, and Nick Coghlan. volume 8.

[60] Jose Antonio Sanchez. ¿cómo aprenden las máquinas? machine learning y sus diferentes tipos, Aug 2020.

[61] Daniel Scheuffler. Code is read more often than it is written, 2016.

[62] scikit-learn developers. Scikit-learn coding guidelines, 2022.

[63] Sunith Shetty. Automl for building simple to complex ml pipelines, Sep 2018.

[64] Alexander Shvets. *Behavioral Design Patterns*, page 381–392. Refactoring.Guru, 3rd edition, 2021.

[65] Jafar Tanha, Maarten Van Someren, and Hamideh Afsarmanesh. Semi-supervised self-training for decision tree classifiers. *International Journal of Machine Learning and Cybernetics*, 8(1):355–370, 2017.

[66] Technovert. Introduction to machine learning, 2020.

[67] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[68] 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.

[69] Isaac Triguero, José A Sáez, Julián Luengo, Salvador García, and Francisco Herrera. On the characterization of noise filters for self-training semi-supervised in nearest neighbor classification. *Neurocomputing*, 132:30–41, 2014.

[70] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[71] Universia. Para qué sirve python: qué es y usos.

[72] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[73] Gang Wang, Fei Wang, Tao Chen, Dit-Yan Yeung, and Frederick H Lochovsky. Solution path for manifold regularized semisupervised classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):308–319, 2011.

[74] Jun Wang, Tony Jebara, and Shih-Fu Chang. Semi-supervised learning using greedy max-cut. *The Journal of Machine Learning Research*, 14(1):771–800, 2013.

[75] Gordon Wilfong. Nearest neighbor problems. *International Journal of Computational Geometry & Applications*, 2(04):383–416, 1992.

[76] D Randall Wilson and Tony R Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

[77] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[78] Ian H Witten, Eibe Frank, Mark A Hall, CJ Pal, and MINING DATA. Practical machine learning tools and techniques. In *DATA MINING*, volume 2, page 4, 2005.

[79] Di Wu, Mingsheng Shang, Xin Luo, Ji Xu, Huyong Yan, Weihui Deng, and Guoyin Wang. Self-training semi-supervised classification based on density peaks of data. *Neurocomputing*, 275:180–191, 2018.

[80] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196, 1995.

[81] Jianhua Zhao and Ning Liu. A safe semi-supervised classification algorithm using multiple classifiers ensemble. *Neural Processing Letters*, 53(4):2603–2616, 2021.

[82] Xueyuan Zhou and Mikhail Belkin. Semi-supervised learning. In *Academic Press Library in Signal Processing*, volume 1, pages 1239–1269. Elsevier, 2014.

[83] Yan Zhou and Sally Goldman. Democratic co-learning. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 594–602. IEEE, 2004.

[84] 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.