UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Semisupervised learning and instance selection methods

Presentado por Daniel Puente Ramírez
en Universidad de Burgos — 11 de enero de 2022
Tutor: 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 título de TFG.

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, 11 de enero de 2022

Vº. Bº. del Tutor:

D. Álvar Arnaiz-González

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

Descripción del contenido del trabajo y del estructura de la memoria y del resto de materiales entregados.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

Conceptos teóricos

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

3.1. Aprendizaje en *machine learning*

En [34] se define *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, cada una de ellas con una aproximación diferente en cuanto al uso de instancias etiquetadas.

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 [24] El objetivo es simple, debe de «aprender» en función del conjunto de datos etiquetado proporcionado como

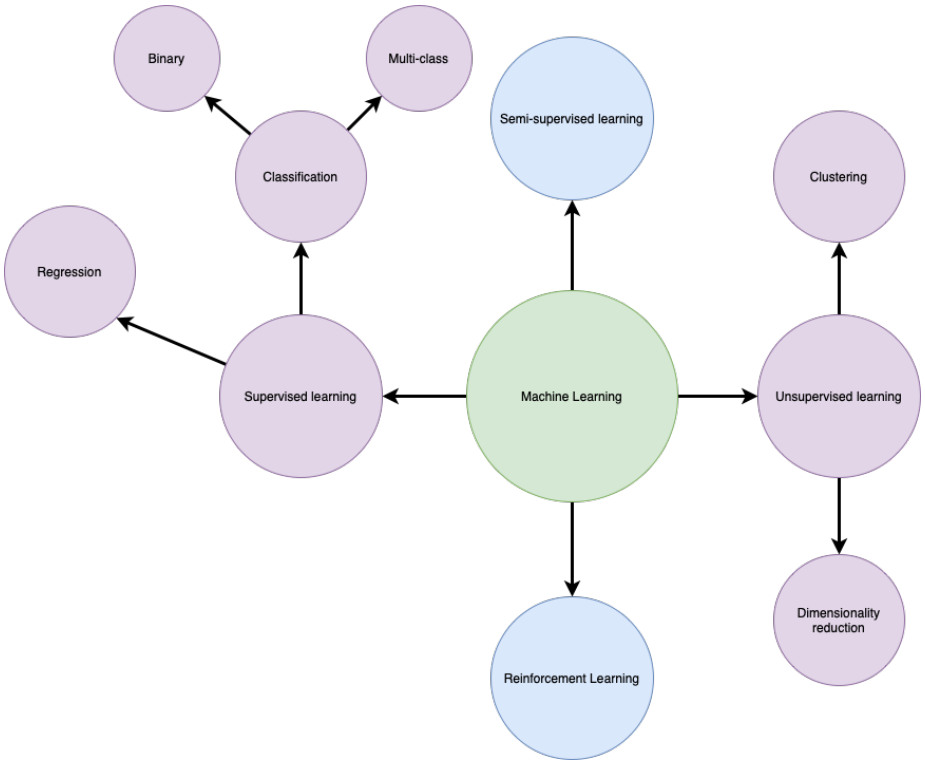

Figura 3.1: *Machine learning overview* [35]

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 [21]

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 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.

   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 utiliza la multi-regresión.

   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 [4], 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 datos.

Principalmente existen dos técnicas de aprendizaje no supervisado.

1. ***Clustering*** [26] 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 hacer con 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.

2. **Reducción de la dimensionalidad.** La clasificación en el aprendizaje automático se basa en atributos o características que tienen los datos, permitiendo 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 [25]

Aprendizaje semi-supervisado

Semi-Supervised Learning según [45], 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, Sección 3.1, o no supervisado, Sección 3.1.

El uso de aprendizaje semi-supervisado se caracteriza por ser más barato 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 largo y costoso. 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 [22] indica las siguientes suposiciones que se dan en el aprendizaje semi-supervisado.

1. **Continuidad.** Se asume que los objetos cercanos entre sí se encontrarán en el mismo *cluster* o grupo de etiquetas.
2. ***Clustering.*** Las instancias son divididas en diferentes grupos discretos, compartiendo todos los elementos de un *cluster* la misma etiqueta.
3. ***Manifold*** o colectores. 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 [36]

3.2. Algoritmos en el aprendizaje semi-sepervisado

A continuación se van a presentar los algoritmos tratados en este trabajo.

Self-training

Los métodos de auto-entrenamiento (*self-training*), son uno de los métodos más sencillos de pseudo-etiquetado que existen Triguero *et al.* [37] Se encuentran basados en un único clasificador el cual utiliza aprendizaje supervisado, el clasificador se encuentra siendo entrenado constantemente con datos etiquetados conocidos y por los que se van conociendo a medida que pasan las iteraciones. Es decir, en el inicio el clasificador es entrenado con los datos etiquetados que se conocen, con ese clasificador se obtienen nuevas instancias etiquetadas y las mejores (con mayor *confidence level*) son añadidas al conjunto de datos etiquetado para volver a entrenar el clasificador [14]

Yarowsky [44] 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* [12]

Co-Training

Blum [5] en 1998 propuso el *Co-Training* para conjuntos de datos compuestos por datos etiquetados y no etiquetados. 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 mayor *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 *bootstraping* a partir de datos incompletos en el marco de la maximización de expectativas [17, 30]

Algorithm 1 *Co-Training.*

Require: 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

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

Ensure: Conjunto etiquetado $S \subset \{L, U\}$

procedure CO-TRAINING(L, U, p, n, k, u)
  $U' \leftarrow U[u]$▷ u instancias aleatorias de U
  for k **do**
    Usar L para entrenar un clasificador h_1 utilizando x
    Usar L para entrenar un clasificador h_2 utilizando y
    h_1 clasifica U'
    h_2 clasifica U'
    Seleccionar las p y n instancias con mayor *confidence level* de cada clasificador
    Añadirlas a L y eliminarlas de U'
    Elegir de forma aleatoria $2p + 2n$ instancias de U y añadirlas a U'
  end for
end procedure

Tri-Training

Zhou [47] en 2005 propuso una modificación sobre el algoritmo de *Co-Training* de Blum [5] el cual requería de dos «vistas significativas». Dasgupta *et al.* [11] 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 significas» 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.

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) [47]

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*[1] 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*.

[1] 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.

Algorithm 2 *Tri-Training.*

Require: Conjunto de entrenamiento L y U de datos etiquetados y no etiquetados, respectivamente
Require: *Learn*: algoritmo de aprendizaje
Ensure: Conjunto etiquetado $S \subset \{L, U\}$

procedure TRI-TRAINING($L, U, Learn$)
  for $i \in \{1.,3\}$ **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 $i \in \{1.,3\}$ **do**
      $L_i \leftarrow \emptyset$
      $update_i \leftarrow$ **False**
      $e_i \leftarrow MeasureError\left(h_j \& h_k\right)\ (j,k \neq i)$
      if $e_i < e'_i$ **then**
        for all $x \in U$ **do**
          if $h_j\left(x\right) = h_k\left(x\right)\ (j,k \neq i)$ **then**
            $L_i \leftarrow L_i \cup \{(x, h_j\left(x\right))\}$
          end if
        end for
        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 $i \in \{1.,3\}$ **do**
      if $update_i =$ **True then**
        $h_i \leftarrow Learn\left(L \cup L_i\right)$
        $e'_i \leftarrow e_i$
        $l'_i \leftarrow |L_i|$
      end if
    end for
  until ningún $h_i\,(i \in \{1.,3\})$ cambie
end procedure

Democratic Co-Training

Zhou [46] 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:

- Solo se poseen unas pocas muestras etiquetadas
- Existe un conjunto de datos de tamaño muy superior, sin etiquetar
- No existen dos conjuntos de atributos independientes y redundantes

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 $n \geq 3$, 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 % $[l, h]$.

Finalmente el algoritmo, ver 4, finaliza 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.

Algorithm 3 *Democratic Co-Learning.*

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

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

Ensure: Conjunto etiquetado $S \subset \{L, U\}$

procedure DEMOCRATIC CO-LEARNING(L, U)
  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
    for all $x \in U$ **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 for
    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

Algorithm 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
for all $x \in (L \cup U)$ **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 for
Predicciones H con Gk para $k = argmax_j \left(\overline{C}_{G_j}\right)$
return H
end procedure

3.3. Minería de datos

Según IBM [13], 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 [9]

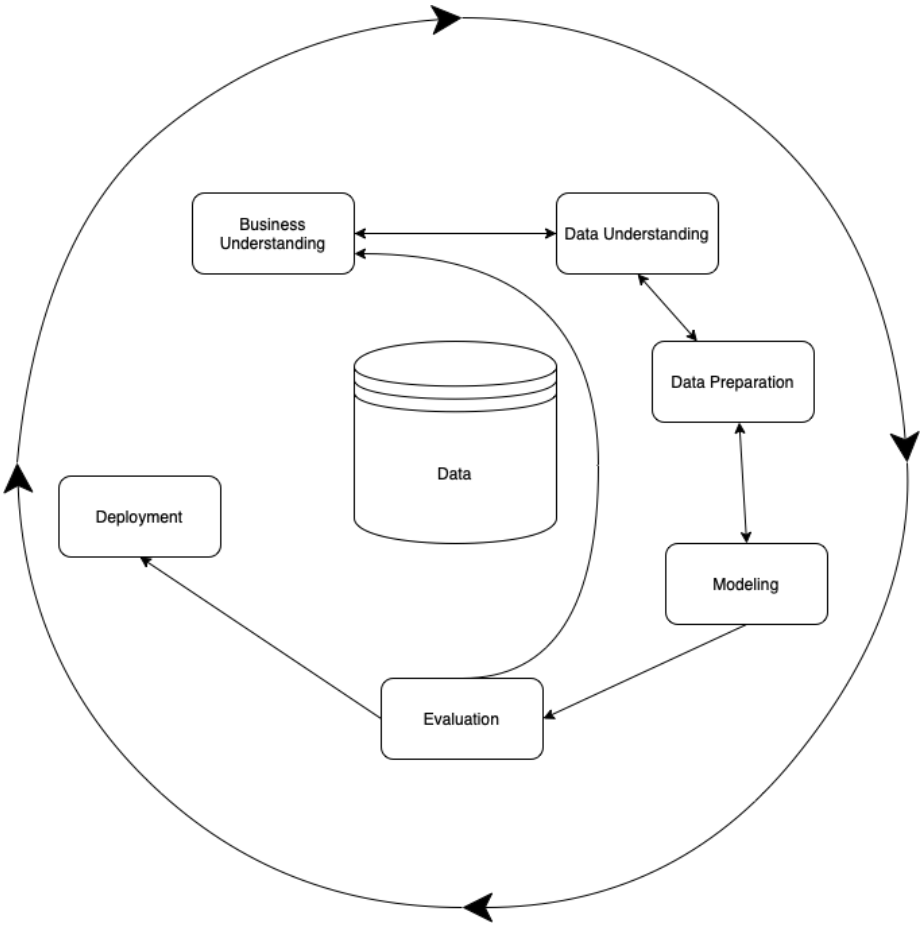

Figura 3.2: Enfoque CRISP de la minería de datos [23]

En [23] 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.

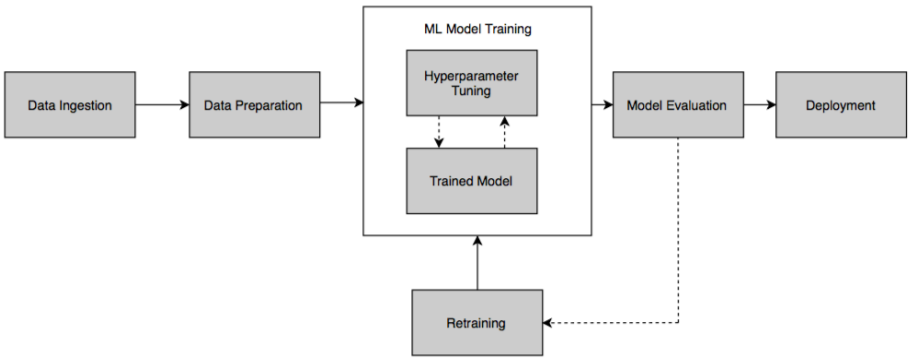

Figura 3.3: *Machine Learning Pipeline* [2]

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 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 [23] 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 como 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.

   Los modelos en función de como resuelvan el problema que se les presenta se pueden clasificar en:

   a) Regresión.

   b) Análisis de asociación.

   c) *Clustering.*

   d) Detección de anomalías.

   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 [9]

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 [28]

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.** 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 [28]

4. **k-vecinos más cercanos. KNN (*k-nearest neighbors*)** [18, 19, 28] 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 [10]

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

Figura 3.4: Proceso de selección de instancias.

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 [27]

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.

Es por ello que un pre-procesado o **filtrado** de las instancias pertenecientes al T original es necesario. En la Figura 3.4 se puede consultar 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.

- **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

Método	Basado en	Referencia
ENN	Clasificación incorrecta	[42]
CNN	Clasificación incorrecta	[20]
RNN	Clasificación incorrecta	[16]
ICF	Alcance y covertura	[6]
MSS	Fronteras de decisión	[3]

Tabla 3.1: Algunos métodos de selección de instancias.

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 [42] (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.

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:

Algorithm 5 Algoritmo de edición de Wilson, *ENN*.

Require: Conjunto de entrenamiento, $X = \{(x_1, y_1)...(x_n, y_n)\}$, k vecinos.
Ensure: Conjunto editado $S \subset X$.

```
1: procedure ENN(X, k)
2:     S ← X
3:     for all x ∈ S do
4:         Find x.N_{1...k+1}, the k+1 nearest neighbors of x ∈ X − {x}
5:         if δ_{k−NN}(x_i) ≠ θ_i then
6:             Remove x of S
7:         end if
8:     end for
9:     return S
10: end procedure
```

El algoritmo de edición de Wilson, ver algoritmo 5 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.

Algoritmo Condensado de Hart

Hart [20] 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.

Algorithm 6 Algoritmo Condensado de Hart, *CNN*.

Require: Conjunto de entrenamiento X
Ensure: Conjunto editado $S \subset X$

```
1: procedure CNN(X)
2:     S ← {x_1}
3:     for all x ∈ X do
4:         if x no se clasifica correctamente usando S then
5:             Añadir x a S
6:             Restart
7:         end if
8:     end for
9: end procedure
```

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 [20], ver algoritmo 6, posee una complejidad 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 Condensado Reducido

Gates [16] en 1972 propuso el algoritmo del conjunto reducido, basado en las reglas *NN*. El algoritmo propuesto es una modificación del algoritmo *CNN*, ver algoritmo 6. 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 7, 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.

Algorithm 7 Algoritmo Condensado Reducido, *RNN*.

Require: Conjunto de entrenamiento X
Ensure: Conjunto editado $S \subset X$

```
1:  procedure RNN(X)
2:      S ← {x_1}
3:      for all x ∈ S do
4:          if x no se clasifica correctamente usando S then
5:              Añadir x a S
6:              Restart
7:          end if
8:      end for
9:      for all x ∈ S do
10:         Remove x de S
11:         if ∃x_i incorrectamente clasificada usando S then
12:             Añadir x a S
13:         end if
14:     end for
15: end procedure
```

Algoritmo *Iterative Case Filtering*

Brighton [6] 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.

Algorithm 8 Algoritmo *Iterative Case Filtering*, *ICF*.

Require: Conjunto de entrenamiento X
Ensure: Conjunto editado $S \subset X$

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

ICF puede describirse como un filtro de borrado de instancias, más que de clasificación, con el objetivo de eliminar aquellas instancias que sean superfluas o que aportern ruido a T [6] 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 8, 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' si 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 Subconjunto Selectivo
Ritter [31] 1975 propone un algoritmo capaz de satisfacer la condición de consistencia del algoritmo condensado de Hart, ver 3.4; 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 reformular 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 [3] 2005 propuso el algoritmo de subconjunto selectivo modificado, *MSS*, como su propio nombre indica, se trata de una modificación del algoritmo propuesto previamente, ver 3.4. Debido a que no se puede garantizar que el algoritmo de Ritter [31] devuelva un subconjunto, S, el cual sea mínimo y consistente, habiendo sido categorizado como un problema NP-Completo [40], 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 9, 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 coimo aquel que contiene el mejor vecino relacionado para cada instancia en el TS. (Mejor $\iff$ Menor distancia a su enemigo más cercano).

Algorithm 9 Algoritmo *Modified Selective Subset, MSS.*

Require: Conjunto de entrenamiento X
Ensure: Conjunto editado $S \subset X$

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

Decremental Reduction Optimization Procedure

Bajo el nombre de *Decremental Reduction Optimization Procedure, DROP* [41] 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 pertence 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 $P.A_{1,\dots,a}$ donde a es el número de asociados de P.

Algorithm 10 Algoritmo *Decremental Reduction Optimization Procedure 3*, *DROP3*.

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

```
1: procedure DROP3(X, k)
2:     T ← Filtrado de ruido(X)
3:     S ← T
4:     Ordenar las instancias de S por la distancia a su enemigo más
   próximo ▷ de mayor a menor
5:     for all x ∈ S do
6:         Encontrar los x.N_{1...k+1}, los k+1 vecniso más cercanos de x en S
7:         Añadir x a la lista de asociados de cada uno de sus vecinos
8:     end for
9:     for all x ∈ S do
10:        with ← # número de asociados de x clasificados correctamente
   con x como vecino
11:        without ← # número de asociados de x clasificados correctamente
   sin x como vecino
12:        if without ≥ with then
13:            Eliminar x de S
14:            for all Asociado a de x do
15:                Eliminar x de la lista de vecinos de a
16:                Encontrar nuevo vecino para a
17:                Añadir a en la lista de asociados del nuevo vecino
18:            end for
19:        end if
20:    end for
21:    return S
22: end procedure
```

Todas las aproximaciones de $DROP\{1 \dots 5\}$ se basan en el mismo algoritmo [41] pero con modificaciones el 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.
- *DROP3*. Lo primero de todo realiza un filtrado de ruido, muy similar a la edición de Wilson, ver en 3.4. Seguidamente aplica el algoritmo *DROP2*, ver algoritmo 10
- *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 $withput > with$ se está tratando con un prototipo el cual no aporta información sobre el conjunto S permitiendo su eliminación.

3.5. Función distancia entre instancias

Una función distancia proporciona 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:

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

- **Distancia del Coseno.** Mide la similitud o distancia entre dos vectores calculando el coseno del ángulo que forman.

  Su fórmula es la siguiente:

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

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 [8], 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.

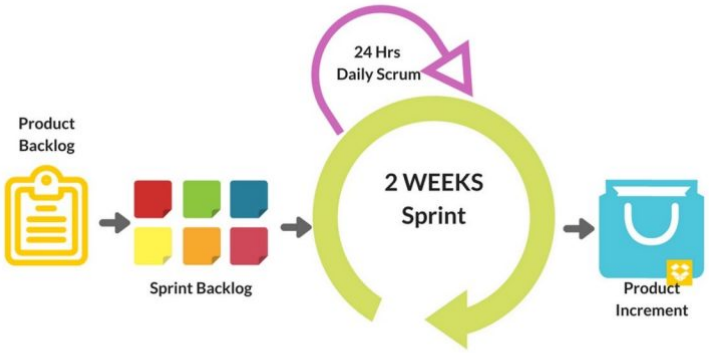

Figura 4.1: Metodología *scrum*.

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* [29]

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* [32]

- ***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 [39], Python es un lenguaje de programación multiplataforma y multiparadigma [2], entre sus principales características se encuentran la legibilidad y limpieza del código. Python es *open source software* por lo que su uso es gratuito [15]

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

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

programación más utilizado para desarrollo de todo tipo de aplicaciones, tanto es así que, según las últimas estadísticas recuperadas por PYPL, ver [7], 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.
- 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* [1] 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.

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 [33]

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.*» (Guido van Rossum, 2001). 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.

4.2. Herramientas

Weka

Desarrollado por la Universidad de Waikato [43] 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 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* [3].

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 estadística del procesamiento del modelo, junto con ello proporciona una herramienta de

[3]Esta es ua 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.

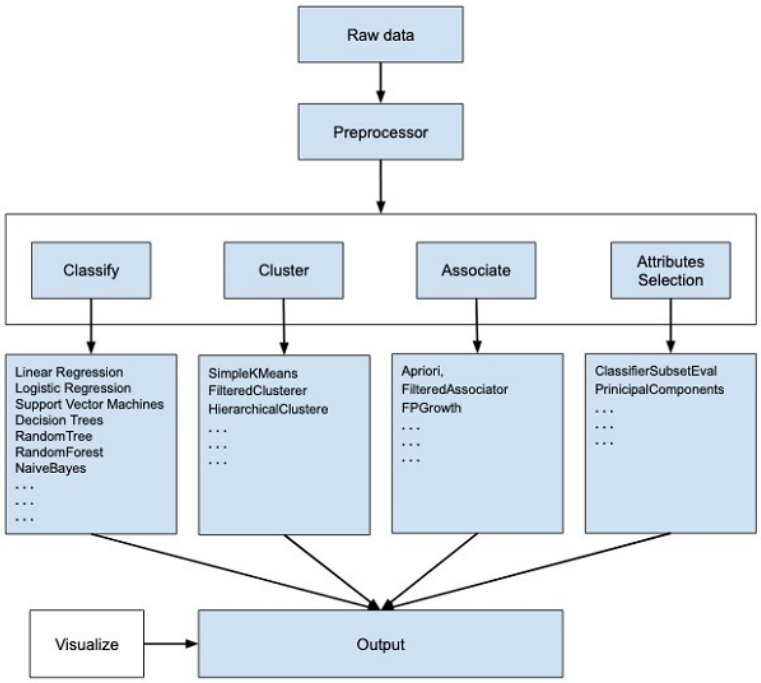

Figura 4.2: Resumen de las funcionalidades de Weka. Imagen recuperada de [38]

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[4] 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.

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

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

Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

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

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

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

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

[12] 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.

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

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

[15] Python Software Foundation. History and license.

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

[17] 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.

[18] 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.

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

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

[21] Mathworks Inc. Supervised learning.

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

[23] 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.

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

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

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

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

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

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

[36] 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.

[37] 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.

[38] tutorialspoint. What is weka?

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

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

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

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

[43] 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.

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

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

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

[47] 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.