UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Aplicación del Aprendizaje Semisupervisado en el descubrimiento de ataques a Sistemas de Recomendación

Presentado por Patricia Hernando Fernández
en Universidad de Burgos — 1 de diciembre 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 la alumna D.ª Patricia Hernando Fernández, con DNI 71362977A, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Aplicación del Aprendizaje Semisupervisado en el descubrimiento de ataques a Sistemas de Recomendación».

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, 1 de diciembre 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 general iii

Índice de figuras iv

Índice de tablas v

Introducción 1
1.1. Preámbulo 1

Objetivos del proyecto 3

Conceptos teóricos 5
3.1. Apendizaje automático 5
3.2. Aprendizaje semisupervisado 5
3.3. *Ensembles* 7
3.4. *Co-forest* 9
3.5. Ataques a sistemas de recomendación 14

Técnicas y herramientas 19

Aspectos relevantes del desarrollo del proyecto 21
5.1. Experimentación 21

Trabajos relacionados 31

Conclusiones y Líneas de trabajo futuras 33

Bibliografía 35

Índice de figuras

3.1. Taxonomía de métodos de aprendizaje semisupervisados. Extraída de *A survey on semi-supervised learning* [4]. 6
3.2. CAMBIAR Representación genérica de un modelo *bagging*. . . . 8
5.1. Gráfica que representa la distribución de los datos. 22
5.2. Gráfica que representa la precisión del modelo en función de la iteración en la que se encuentre. 23
5.3. Gráfica que representa, para cada conjunto de datos, la precisión media del modelo en función de la iteración en la que se encuentre junto con la desviación típica para 10 experimentos. 24
5.4. Gráfica que representa la evolución del tiempo de entrenamiento en función del número de instancias utilizadas. 25
5.5. Gráfica que representa la precisión media del modelo en función del porcentaje de datos utilizados para el entrenamiento. 26
5.6. Gráfica que representa, para cada conjunto de datos, la precisión media del modelo y su desviación en función del porcentaje del conjunto utilizado para el entrenamiento. 27

Índice de tablas

3.1. Descripción de los ataques básicos. [11] 15
3.2. Descripción de los ataques con poco conocimiento del sistema. . 16
3.3. Descripción de las estrategias de ofuscación. 17
5.1. Descripción de los *datasets* utilizados para probar el *co-forest*. . 22
5.2. Tabla resumen con el diseño del experimento. 28
5.3. Comparativa entre la exactitud del *co-forest* de KEEL y el propio sobre el conjunto de *test*. 29

Introducción

1.1. Preámbulo

A diferencia de unas décadas atrás, la sociedad actual está gobernada por los datos. La transición a la era de la información puede ser compleja para determinados colectivos y, consecuentemente, diversos sistemas auxiliares han sido desarrollados con el fin de resumir información y facilitar la toma de decisiones. Entre ellos se encuentran los sistemas de recomendación, que son herramientas que pretenden realizar sugerencias de objetos que pueden resultar interesantes para un determinado perfil.

Económicamente, este tipo de algoritmo es un claro objeto de interés, puesto que puede influir en la toma de decisiones de los compradores y hacer que se inclinen por un determinado producto (por ejemplo, el que tenga una mejor valoración). Los atacantes conocen esta situación y manipulan estas herramientas mediante el uso de perfiles falsos con el fin de beneficiar sus productos o perjudicar los de la competencia.

Este proyecto de investigación pretende explorar cómo el aprendizaje semisupervisado puede ayudar a detectar los ataques a sistemas de recomendación, diferenciando entre perfiles genuinos e inyectados, además de comprobar la veracidad de los planteados por otros investigadores.

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

Se sintetizarán a continuación algunos de los conceptos teóricos más relevantes para la correcta comprensión del documento.

3.1. Apendizaje automático

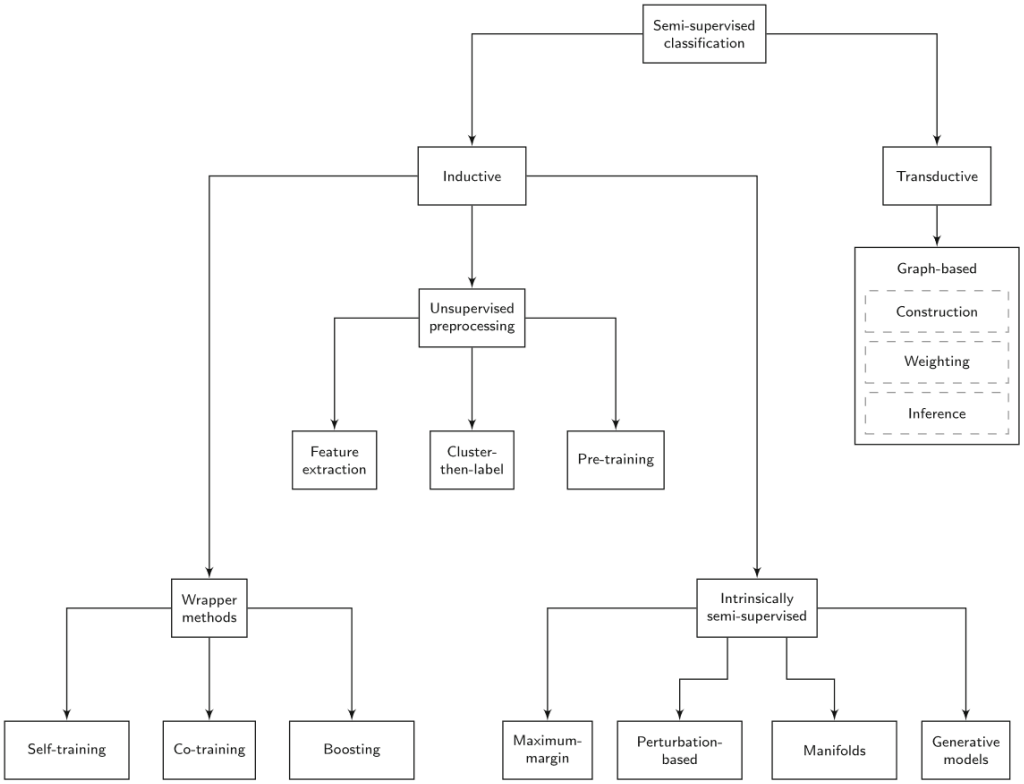
Se denomina aprendizaje automático a aquella rama de la inteligencia artificial cuyo objetivo es desarrollar métodos que permitan que un algoritmo mejore su rendimiento mediante la experiencia y procesado de datos. Consecuentemente, los modelos entrenados realizarán predicciones cada vez más precisas como resultado del algoritmo implementado.

Dentro del aprendizaje automático se diferencian tres grandes grupos en función del tipo de entrada que sea consumida: el aprendizaje supervisado (datos etiquetados), el no supervisado (datos no etiquetados) y el semisupervisado (datos etiquetados y no etiquetados), siendo esta última categoría objeto de estudio en este proyecto de investigación.

3.2. Aprendizaje semisupervisado

Como se ha mencionado anteriormente, se denomina aprendizaje semisupervisado a aquel conjunto de algoritmos que utiliza datos etiquetados y no etiquetados para realizar tareas de aprendizaje. Inicialmente, se pueden diferenciar dos categorías [4]: los métodos inductivos, cuyo objetivo principal es construir un clasificador que genere predicciones para cualquier entrada y los métodos transductivos, cuyo poder de predicción está limitado a los objetos utilizados en la fase de entrenamiento.

Figura 3.1: Taxonomía de métodos de aprendizaje semisupervisados. Extraída de *A survey on semi-supervised learning* [4].



Prescindiendo de los métodos transductivos por ser menos versátiles y útiles en nuestro propósito, los métodos inductivos se subdividen en tres grupos [4]: *wrapper methods* (o métodos de envoltura), *unsupervised preprocessing* y *intrinsically semi-supervised*, siendo materia de estudio los métodos de envoltura.

Métodos de envoltura

Estos modelos utilizan uno o más clasificadores que son entrenados iterativamente con los datos etiquetados de entrada, además de con datos pseudo-etiquetados. Se denomina pseudo-etiquetado a aquellos datos que inicialmente no estaban etiquetados, pero acabaron estándolo por iteraciones previas de los clasificadores.

Consecuentemente, el procedimiento consta de dos fases que se repiten en cada iteración: el entrenamiento y el pseudo-etiquetado. Durante el entrenamiento, los clasificadores se alimentan de datos etiquetados (o pseudo-

etiquetados). En la fase de pseudo-etiquetado, se utilizan datos no etiquetados para que sean procesados por los clasificadores previamente entrenados.

Dentro de esta categoría, se pueden diferenciar tres grandes grupos: *self-training*, que utilizan únicamente un clasificador, *co-training* [4], que utilizan más de uno y los *pseudo-labelled boosting methods*, que construyen clasificadores individuales que se alimentan de las predicciones más fiables. Se estudiará más en profundidad los métodos *co-training*.

Co-training

En estos algoritmos, varios clasificadores son entrenados iterativamente utilizando datos etiquetados y añadiendo las predicciones (resultados) más confiables al conjunto para ser utilizadas en las siguientes iteraciones [4]. Para que los clasificadores sean capaces de generar información distinta, generalmente se divide el conjunto de entrada según alguna característica (no siendo estrictamente necesario). El *co-forest*, algoritmo protagonista de este proyecto, pertenece a esta categoría.

TO-DO: Añadir información sobre las dos vistas independientes del co-training (otros métodos no tienen esta limitación)

3.3. *Ensembles*

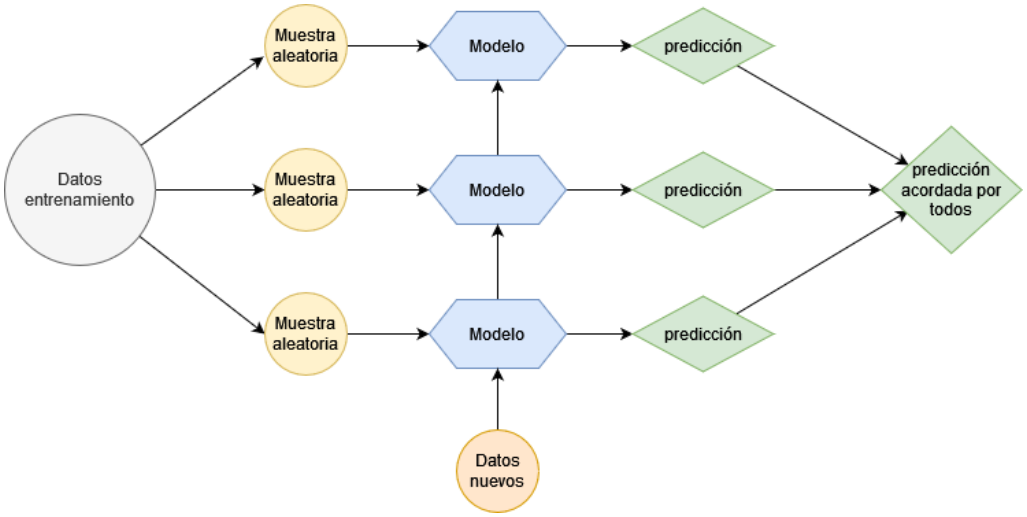
Se define *ensemble* como un conjunto de modelos de *machine learning* donde cada estimador base genera una predicción individual que se combina con el resto para generar una salida única [5]. Los *ensembles* pueden ser de clasificación o regresión, siendo los mostrados a continuación los más comunes.

Bagging

Este modelo se compone de un conjunto de estimadores base entrenados en paralelo. Por lo tanto, el resultado es el promedio (o más popular) de las salidas de los modelos simples. Para entrenar cada estimador base se utiliza *bootstrapping* [8], que es una técnica de muestreo que genera un subconjunto de datos seleccionando aleatoriamente muestras de un conjunto mayor permitiendo la repetición. Es decir, los clasificadores son entrenados con un subconjunto aleatorio del total de los datos etiquetados como se ilustra en la figura 3.2. Esta técnica puede ser utilizada con o sin reemplazo, entendiendo que el muestreo con reemplazo se produce cuando cada una de

las unidades maestrales seleccionada es devuelta a la población total antes de extraer la siguiente (puede repetirse).

Figura 3.2: CAMBIAR Representación genérica de un modelo *bagging*.



Random forest

Es uno de los métodos de *bagging* más populares, que obtiene resultados certeros debido a la aleatoriedad introducida [5]. Como clasificadores base, utiliza un conjunto de árboles de decisión, y cuando se devuelve una etiqueta, es el resultado de una votación realizada por todos los árboles del conjunto. Además de utilizar *bootstrapping*, en el entrenamiento individual de cada árbol, únicamente se «ven» algunos atributos (no el total) para introducir mayor aleatoriedad. Es decir, en cada nodo del árbol solo se tienen en cuenta ciertas características seleccionadas aleatoriamente.

Boosting

Si el anterior modelo utiliza los clasificadores base «en paralelo», este lo haría «en serie» [8]. Es decir, hay un orden secuencial: los estimadores dependen del resultado anterior y tratan de compensar el error que se haya podido cometer.

El entrenamiento se focaliza, por lo tanto, en torno a las instancias que hayan fallado los clasificadores previos. Esto se consigue dando más peso a aquellas muestras mal clasificadas (en problemas de regresión, por ejemplo, las predicciones con un mayor error cuadrático medio tendrán más peso para el siguiente modelo).

3.4. *Co-forest*

Descripción

El denominado *co-forest* [5] es un método de clasificación para aprendizaje semisupervisado (más concretamente, de envoltura) que permite construir un *ensemble* de árboles de decisión (clasificación). Este método está basado en el *random forest* y se podría considerar su «versión semisupervisada».

Que sea un algoritmo semisupervisado implica que, durante la fase de entrenamiento, además de utilizar el conjunto de datos etiquetados (L), se utilicen también aquellas pseudo-etiquetas de las predicciones con mejor confianza del conjunto de datos no etiquetados (U).

El proceso de entrenamiento se realiza iterativamente [8], y en cada una de estas repeticiones, se examina cada árbol y se entrena individualmente con un subconjunto de L (L_i) y con un subconjunto de U ($L'_{i,t}$) formado por aquellas pseudo-etiquetas que, además de pertenecer a la submuestra (aleatoria), posean un nivel de confianza superior a un umbral definido por el usuario (θ). Quienes estiman esta confianza de predicción para las muestras seleccionadas son el conjunto de todos los árboles que forman el *ensemble* menos el árbol que está siendo entrenado individualmente (de ahora en adelante *concomitant ensemble* o conjunto concomitante).

El criterio de parada de la fase de entrenamiento del *co-forest* es el *out-of-bag error* [11] (de ahora en adelante, OOBE). Se define OOBE como el error que comete el conjunto concomitante de un árbol para el total de los datos etiquetados. Es decir, el número de árboles que aciertan la etiqueta entre el número de árboles que votan teniendo en cuenta que, para calcular el OOBE del total de los datos etiquetados, sólo votan aquellos árboles que no hayan utilizado la muestra concreta de L para su entrenamiento.

Algoritmo

Variables utilizadas

Para facilitar la comprensión del pseudocódigo, se definen a continuación algunas de las variables utilizadas:

- n: número total de árboles del ensemble.
- h_i: árbol i-ésimo del conjunto.

- H_i: *concomitant ensemble* o conjunto concomitante del árbol i-ésimo (todos los árboles del *co-forest* menos él).
- x_j: muestra j-ésima (generalmente sin etiqueta).
- $H_i(x_j)$: etiqueta asignada por H_i a la muestra j-ésima.
- $w_{i,t,j}$: confianza de predicción (individual) calculada para una muestra x_j por H_i en la iteración t. Se define confianza como el grado de acuerdo que existe en una votación.
- W: sumatorio de las confianzas de predicción individuales de un conjunto $W = \sum w_{i,t,j}$.
- L: conjunto de datos etiquetados utilizados durante el entrenamiento.
- L_i: subconjunto obtenido tras aplicar *bootstrapping* a L que se utiliza para entrenar el árbol h_i.
- U: conjunto de datos no etiquetados utilizados durante el entrenamiento.
- $U'_{i,t}$: subconjunto aleatorio de U cuya W es menor que $Wmax$ (consultar ecuación 3.5).
- $L'_{i,t}$: subconjunto formado por aquellas muestras de $U'_{i,t}$ cuya confianza de predicción es superior a θ.
- θ: nivel de confianza mínimo que tiene que tener el clasificador al predecir la etiqueta de una muestra de U para ser utilizada durante el entrenamiento de un árbol.
- $\hat{e}_{i,t}$: OOBE cometido por H_i en L en el instante de tiempo t.

Pseudocódigo

Se facilita a continuación la versión del *co-forest* de la tesis de Engelen [8], basada en la original [5], pero introduciendo los cambios observables en la sección «Discusión de los parámetros del algoritmo».

Fases

En primer lugar, se ha de construir un *random forest* de n árboles. Para introducir aleatoriedad, cada uno de esos árboles es entrenado utilizando un subconjunto aleatorio de L. Es decir, un subconjunto

Algoritmo 1: *Co-Forest*

Input: Conjunto de datos etiquetados $L\{(x_i, y_i)\}_{i=1}^{l}$, conjunto de datos no etiquetados $U\{x_i\}_{i=l+1}^{k}$, número de árboles n y umbral de confianza θ

Input: *Ensemble* de árboles entrenado H.

for i = *0, ...,* $n-1$ **do**
  $L_i \leftarrow$ Bootstrap(L)
  h_i = EntrenarArbol(L_i)
  $\hat{e}_{i,t} \leftarrow 0{,}5$
  $W_{i,0} \leftarrow min(\frac{1}{10}|\mathrm{U}|, 100)$
end for
$t \leftarrow 0$
while *Algún árbol reciba pseudo-etiquetas* **do**
  $t \leftarrow t+1$
  for i = *0, ...,* $n-1$ **do**
    $\hat{e}_{i,t} \leftarrow$ OOBE(H_i, L)
    $L'_{i,t} \leftarrow \emptyset$
    if $\hat{e}_{i,t} < \hat{e}_{i,t-1}$ **then**
      $W_{max} = \hat{e}_{i,t-1} W_{i,t-1} / \hat{e}_{i,t}$
      $U'_{i,t} \leftarrow$ Submuestrear(U, H_i, W_{max})
      $W_{i,t} \leftarrow 0$
      foreach $x_j \in U'_{i,t}$ **do**
        if *Confianza*$(H_i, x_j) > \theta$ **then**
          $L'_{i,t} \leftarrow L'_{i,t} \cup x_j, H_i(x_j)$
          $W_{i,t} \leftarrow W_{i,t} + Confidence(H_i, x_j)$
        end if
      end foreach
    end if
  end for
  for i = *0, ...,* $n-1$ **do**
    if $(e_{i,t} * W_{i,t} < e_{i,t-1} * W_{i,t-1})$ **then**
      h_i = EntrenarArbol($L_i \cup L'_{i,t}$)
    end if
  end for
end while
return H

obtenido tras aplicar *bootstrap* a L. Otro parámetro a tener en cuenta es el número de atributos a considerar en cada árbol de decisión. Por defecto, se ha establecido este valor a la raíz cuadrada del total. Sin embargo, también podría ser el log_2 del total (heurística de Breiman [8]) o el total, entre otros.

La segunda fase es entrenar el *random forest* de manera semisupervisada e iterativa hasta que se cumpla el criterio de parada (como se ha mencionado anteriormente, basado en el OOBE). Para ello, se calcula el OOBE de un árbol para una iteración. Si es superior al anterior, se considera que el rendimiento ha empeorado para ese árbol. El algoritmo finaliza cuando todos los árboles empeoran su rendimiento en una determinada iteración.

Por el contrario, en caso de que se haya mejorado, se toma una submuestra de U para pseudo-etiquetar (evidentemente, distinta para cada árbol). Posteriormente se examina cada una de las muestras que forman $U'_{i,t}$, y en caso de que el nivel de confianza supere el umbral, se selecciona esa muestra para el entrenamiento (pasa a formar parte de $L'_{i,t}$).

El último paso sería reentrenar los árboles que hayan cambiado con su propio conjunto inicial de datos etiquetados unido a las pseudo-etiquetas generadas en la correspondiente iteración ($L_i \cup L'_{i,t}$). Es decir, en cada iteración se considera U al completo para poder generar la muestra con la que se pseudo-etiquete, y las anteriores pseudo-etiquetas para un árbol en concreto son descartadas [8].

Tratamiento del ruido y teoría de errores

Como se señala en el *paper* de Zhou y Duan [11], de acuerdo con el trabajo de Angluin y Laird [1], si el tamaño de los datos utilizados en el entrenamiento (m), la tasa de ruido (η), el error de la hipótesis en el peor caso (ϵ) y una constante (c) cumplen la relación de la ecuación 3.1, entonces la hipótesis aprendida por el árbol h_i (que minimiza el desacuerdo en un conjunto de muestras de entrenamiento con ruido) converge a la hipótesis verdadera.

$$m = \frac{c}{\epsilon^2(1-2\eta)^2} \tag{3.1}$$

De acuerdo con [11], se puede obtener la función de utilidad mostrada en la ecuación 3.2 operando en la expresión 3.1.

$$f = \frac{c}{\epsilon^2} = m(1-2\eta)^2 \tag{3.2}$$

Como se ha mostrado en el pseudocódigo, en la iteración i-ésima un determinado árbol se entrena con sus datos etiquetados L_i y un conjunto de pseudo-etiquetas $L'_{i,t}$. Si se considera que el OOBE cometido en L por H_i es $\hat{e}_{i,t}$, entonces se puede estimar que el número de pseudo-etiquetas erróneas en $L'_{i,t}$ equivale a $\hat{e}_{i,t} \times W_{i,t}$ (se recuerda al lector que $W_{i,t}$ es el sumatorio de la confianza de predicción (grado de acuerdo) de H_i en cada muestra de $L'_{i,t}$). Por lo tanto, la tasa de ruido que se encuentra en $L_i \cup L'_{i,t}$ es la estimada por la ecuación 3.3, donde W_0 y η_0 son los parámetros correspondientes a L.

$$\eta = \frac{\eta_0 W_0 + \hat{e}_{i,t} W_{i,t}}{W_0 + W_{i,t}} \tag{3.3}$$

De acuerdo con la ecuación 3.2, la función de utilidad f es inversamente proporcional a ϵ^2. Por lo tanto, si se quiere reducir el error cometido, se debe aumentar la utilidad de cada árbol en cada iteración [11]. Consecuentemente, se debe cumplir la ecuación 3.4.

$$\frac{\hat{e}_{i,t}}{\hat{e}_{i,t-1}} < \frac{W_{i,t-1}}{W_{i,t}} < 1 \tag{3.4}$$

Discusión acerca de los parámetros del algoritmo

Intuitivamente se puede deducir la ecuación 3.4, ya que el error debe disminuir y la confianza de predicción aumentar con cada iteración. Sin embargo, aunque esto se cumpla, puede ser que se deje de cumplir que $\hat{e}_{i,t} W_{i,t} < \hat{e}_{i,t-1} W_{i,t-1}$, ya que puede ocurrir que $W_{i,t} >>> W_{i,t-1}$. Por este motivo y para cumplir con lo expuesto en 3.4, se limita W_{max} como se muestra en la ecuación 3.5 al realizar el muestreo de U en el algoritmo.

$$W_{max} = \frac{\hat{e}_{i,t-1} W_{i,t-1}}{\hat{e}_{i,t}} > W_{i,t} \tag{3.5}$$

El algoritmo original propuesto por [5] y el utilizado en [11] dejan, sin embargo, una cuestión pendiente. Como se puede observar en la ecuación 3.5, W_{max} requiere para calcularse tanto el OOBE como la W obtenida en la iteración anterior, y ambos autores inician W a 0. Esto resulta en que en la primera iteración $W_{max} = 0$ y, por lo tanto, evita que se realice un muestreo de U para pseudo-etiquetar (pararía el algoritmo). En su tesis, Engelen [8]

propone solucionar este problema iniciando $W = min(\frac{1}{10}|U|, 100)$, aunque destaca que imponer esta constante hace que el impacto de los datos sin etiquetar en el algoritmo dependa profundamente del tamaño del *dataset*.

3.5. Ataques a sistemas de recomendación

Los ataques a los sistemas de recomendación (generalmene denominados *shilling attacks* [6] o *profile injection attack* [9]) tienen como objetivo manipular las sugerencias que propone un determinado algoritmo para conseguir que un cliente se incline hacia un elemento deseado. Esta alteración del sistema se consigue inyectando perfiles falsos.

Múltiples estudios se han centrado en formalizar las características de estos ataques con el fin de detectarlos. Entre ellas se encuentran [6]:

- **Intención:** normalmente, se pretende manipular la opinión general acerca de un elemento (ya sea para bien o para mal). Según el objetivo se pueden diferenciar dos tipos de ataques: ***push attacks***, que pretenden hacer un objeto más atractivo o ***nuck attacks***, cuya intención es la contraria. En caso de que el atacante no busque alterar la opinión acerca de un producto sino restar credibilidad a un sistema (mediante valoraciones aleatorias), se habla de ***random vandalism*** [2].
- **Fuerza:** la calidad de los ataques se mide teniendo en cuenta el **tamaño del relleno** (número de valoraciones asignadas a un perfil atacante, que suele rondar entre el 1 y el 20 % del total de los ítems [6]) y el **tamaño del ataque** (número de perfiles inyectados en el sistema, rondando entre el 1 y el 15 %).
- **Coste:** se distinguen dos tipos: ***knowledge-cost***, que hace referencia al coste de construir perfiles y ***deployment-cost***, que es el número de perfiles que se deben inyectar para conseguir un ataque efectivo [9].

Tipos de ataques

En la actualidad se distinguen multitud de ataques distintos. Con el fin de formalizarlos matemáticamente, se han establecido ciertos conjuntos de interés dependiendo de los ítems que contengan [11].

- I_S**:** conjunto de ítems seleccionados para recibir un tratamiento especial (puede ser vacío).

Modelo	$\mathbf{I_S}$	**Valoración** $\mathbf{I_F}$	$\mathbf{I_0}$	**Valoración** $\mathbf{I_t}$
Random	$\emptyset$	Aleatoria siguiendo una distribución normal definida por todas las valoraciones para todos los ítems del sistema $\mathcal{N}(\mu, \sigma)$.	$\emptyset$	máxima o mínima
Average	$\emptyset$	Aleatoria siguiendo una distribución normal definida por las otras valoraciones para ese ítem en concreto $\mathcal{N}(\mu_i, \sigma_i)$.	$\emptyset$	máxima o mínima

Tabla 3.1: Descripción de los ataques básicos. [11]

- I_F**:** conjunto de ítems seleccionados para «rellenar».
- I_0**:** conjunto de ítems pertenecientes al sistema de recomendación sin valorar.
- I_t**:** conjunto de ítems objetivo.

Ataques básicos

Se distinguen dos tipos: *random attack* y *average attack* [6]. Ambos tienen parámetros y características muy similares como se muestra en la tabla 3.1. La principal diferencia reside en que el *average attack* es mucho más potente debido a que cuenta con mayor información acerca del sistema: las valoraciones a los ítems de relleno siguen una distribución $\mathcal{N}(\mu_i, \sigma_i)$, en lugar de $\mathcal{N}(\mu, \sigma)$. Es decir, la valoración para un determinado ítem se adecúa a la distribución concreta de ese ítem en lugar de a la de todo el *dataset*.

Ataques con poco conocimiento del sistema

Los más populares son *bandwagon attack* (o *popular attack*) y *segment attack*. Sus principales rasgos se ilustran en la tabla 3.2.

La principal característica del *bandwagon attack* es que el conjunto I_S ya no está vacío, sino que contiene algunos de los ítems más populares de la base de datos [11]. Estos ítems recibirán también la máxima puntuación posible, de forma que ya no sólo se puntúa el conjunto objetivo. Existe una variante de este ataque llamada *reverse bandwagon attack*, cuyo objetivo es

Modelo	$\mathbf{I_S}$	**Valoración** $\mathbf{I_F}$	$\mathbf{I_0}$	**Valoración** $\mathbf{I_t}$
Bandwagon (average)	Ítems populares (valoración máxima) o ítems desfavorecidos (puntuación mínima) (reverse)	Aleatoria siguiendo una distribución normal definida por las otras valoraciones para ese ítem en concreto $\mathcal{N}(\mu_i, \sigma_i)$.	$\emptyset$	máxima o mínima (reverse)
Bandwagon (random)	Ítems populares (valoración máxima) o ítems desfavorecidos (puntuación mínima) (reverse)	Aleatoria siguiendo una distribución normal definida por todas las valoraciones para todos los ítems del sistema $\mathcal{N}(\mu, \sigma)$.	$\emptyset$	máxima o mínima (reverse)

Tabla 3.2: Descripción de los ataques con poco conocimiento del sistema.

hacer *nuke*. De esta forma, I_S contiene los ítems menos populares y reciben la puntuación mínima (junto con I_t).

En el *segment attack*, se realiza un pequeño «estudio de mercado» y se introduce en I_S ítems en los que estaría interesado un usuario que fuese a valorar también I_t (de forma que el ataque es más realista).

Ataques con gran conocimiento del sistema

Este tipo de ataques resulta menos relevante que los anteriores debido a la dificultad de su ejecución. En la mayoría de los casos, se necesita una gran cantidad de información, siendo poco realista que se produzca una situación de estas características en la realidad.

Por ejemplo, el llamado *perfect knowledge attack* [9] basa su efectividad en reproducir la distribución exacta de la base de datos real (exceptuándo los ítems objetivos). El *sampling attack* construye los perfiles a inyectar basándose en una muestra de perfiles reales [6].

Como se puede intuir, conocer datos estadísticos exactos sobre una base de datos o metadatos asociados a perfiles de usuarios es poco realista (cada vez menos debido a las mayores medidas de seguridad) y por lo tanto estos ataques resultan meramente teóricos.

Modelo	**Estrategia de ofuscación**
Noise Injection	$\forall i \in I_F \cup I_S : R_i = r_i + aleatorio * \alpha$
Target Shifting	$\forall i \in I_F \cup I_S : R_i = r_i; I_T : r_{max} - 1$ o $r_{min} + 1$
AOP	I_F escogido del top ítems más populares.

Tabla 3.3: Descripción de las estrategias de ofuscación.

Ataques ofuscados

Los ataques ofuscados [6] se basan en intentar «camuflar» los perfiles inyectados haciéndolos pasar por reales. Algunas de las características de su implementación se pueden consultar en la tabla 3.3.

El ataque de *noise injection* introduce a los conjuntos I_S e I_F un «ruido» (número aleatorio que sigue una distribución Gaussiana) multiplicado por una constante α. *Target shifting* incrementa (o decrementa) en una unidad la valoración de I_t con el fin de crear diferencias entre ataques similares sin influir excesivamente el resultado y el *Average over popular items (AOP)* pretende ofuscar el *average attack* cambiando la forma de selección de I_F (en lugar de seleccionar los ítems del conjunto total de la colección, se seleccionan los X % ítems más populares).

Otros tipos de ataques

Además de los ataques previamente ilustrados, existen otros con objetivos más diversos o estrategias distintas. El anteriormente mencionado *random vandalism* (cuya intención únicamente es degradar la calidad del recomendador para causar descontento entre los usuarios) pertenece a esta categoría. Se pueden distinguir, además, ataques basados en copiar comportamientos de usuarios influyentes (modelo *PUA* (*Power User Attack*)) o ítems poderosos (modelo *PIA* (*Power Item Attack*)) [6]. Sin embargo, son menos populares.

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

Aspectos relevantes del desarrollo del proyecto

5.1. Experimentación

Co-Forest

Para evaluar la calidad del método, se han realizado distintos experimentos con los *dataset* incluídos en la librería *SKlearn* dedicados a la clasificación. Estos son los mostrados en la tabla 5.1. El parámetro n representa el número de instancias que contiene un determinado conjunto de datos, mientras que el parámetro m muestra el número de atributos que contiene cada una de esas instancias.

Se han realizado diferentes experimentos: algunos relacionados con la fase de entrenamiento y otros con el algoritmo una vez finalizado.

Fase de entrenamiento

Como se ha desarrollado en los conceptos teóricos, la fase de entrenamiento en el *co-forest* es iterativa, y finaliza cuando ningún árbol recibe nuevas pseudo-etiquetas que puedan cambiar su comportamiento (en la fase de re-entrenamiento).

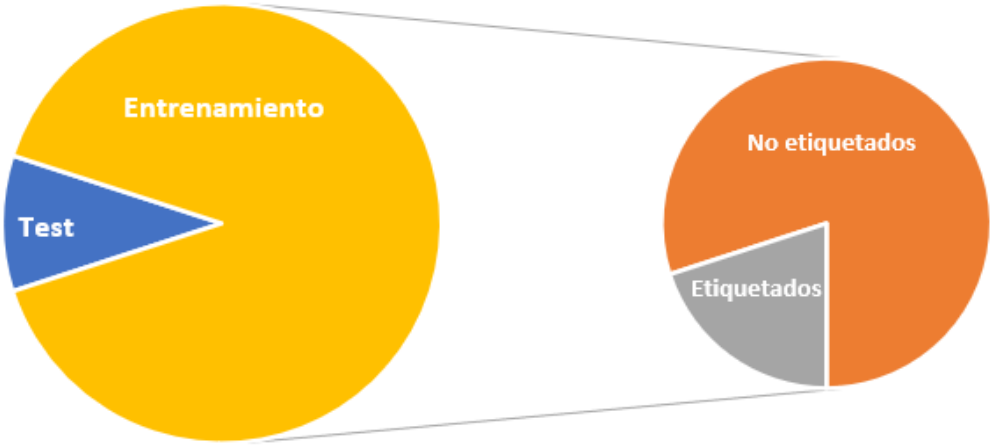
Se ha querido estudiar la evolución de la precisión del algoritmo durante la fase de entrenamiento para los cuatro conjuntos de datos definidos en la tabla 5.1. Para ello, se ha realizado una gráfica comprobando cómo evoluciona la precisión del algoritmo en función de la iteración en la que se encuentre.

Nombre	**Descripción**	**Clases**	n	m
Iris	Conjunto de instancias pertenecientes a diferentes tipos de plantas de la especie Iris.	3	150	4
Dígitos	Conjunto de instancias que representan una imagen de 8x8 perteneciente a un dígito.	10	1797	64
Vino	Conjunto de instancias pertenecientes tres clases de vino con sus parámetros estimados mediante análisis químico.	3	178	13
Cáncer de Mama	Conjunto de instancias que representan parámetros de distintas mujeres que pueden padecer o no cáncer (clasificación binaria).	2	569	30

Tabla 5.1: Descripción de los *datasets* utilizados para probar el *co-forest*.

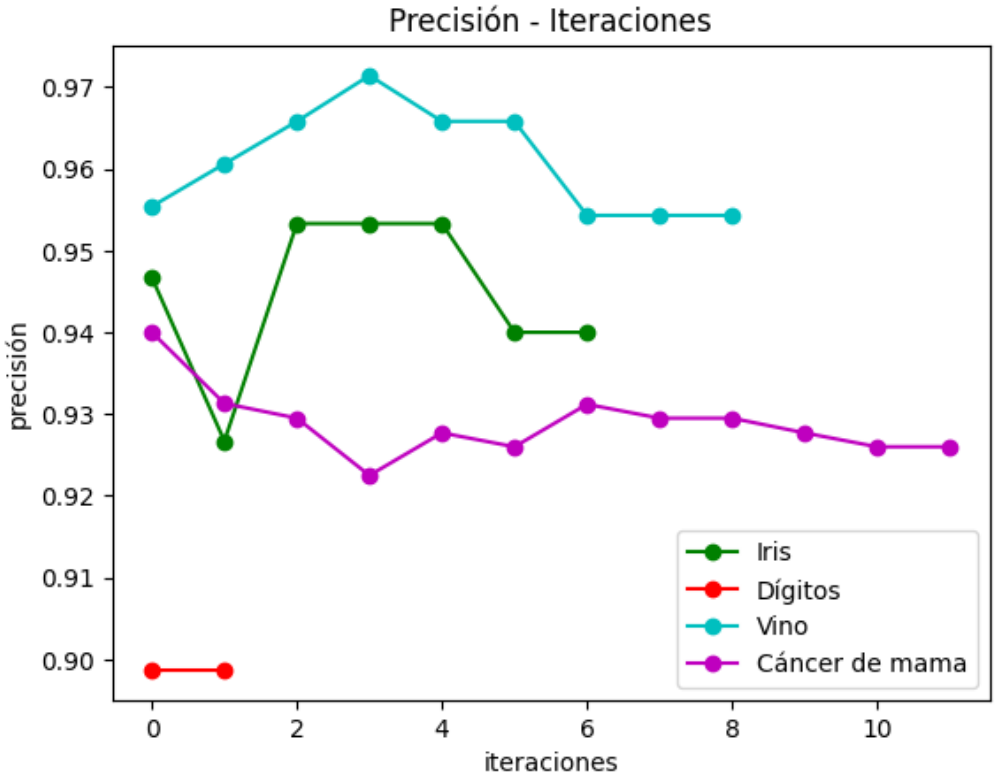
Para garantizar que los resultados obtenidos no son producto de una partición concreta de los datos, se ha realizado validación cruzada (con 10 particiones). Por lo tanto, el porcentaje de datos utilizados para el entrenamiento es el 90 % del total. Los datos de entrenamiento, a su vez, se dividen en etiquetados y no etiquetados. En este caso, el 20 % representa los datos etiquetados, y el 80 % los no etiquetados, como se puede observar en la imagen 5.1. Se han utilizado 20 árboles.

Figura 5.1: Gráfica que representa la distribución de los datos.



La precisión media obtenida se puede ver representada en la gráfica 5.2. Es destacable que, dependiendo de los datos que se utilicen para entrenar el *co-forest*, puede variar el número de iteraciones que se necesiten (incluso dentro de un mismo *dataset*). Por ello, siempre se representa el número máximo de iteraciones realizadas, y para no deformar la media, se ha considerado que el valor de las iteraciones inexistentes es el mismo que el valor obtenido en la última iteración (ya que si, el algoritmo siguiese, el resultado devuelto sería igual debido a que no se re-entrenaría ningún árbol).

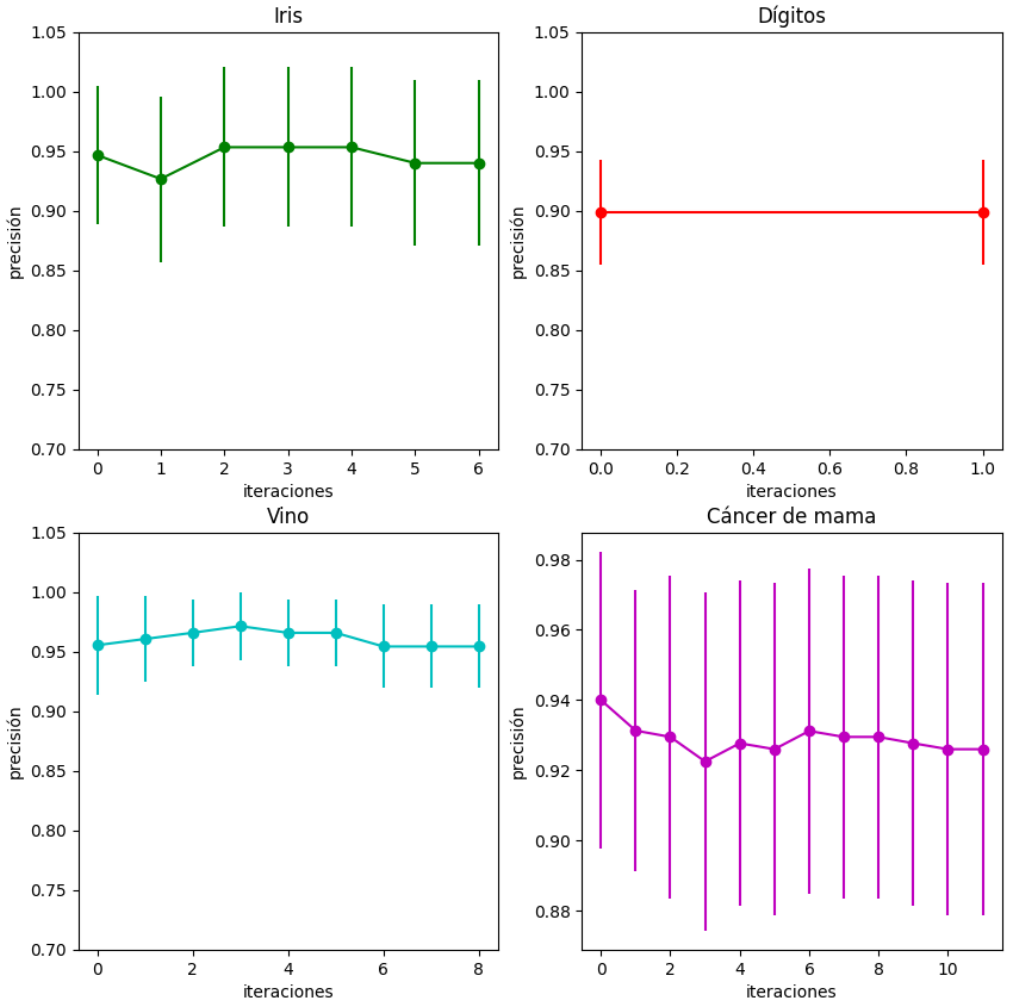
Figura 5.2: Gráfica que representa la precisión del modelo en función de la iteración en la que se encuentre.



Como se puede comprobar, en general los resultados obtenidos no son satisfactorios debido a que la precisión alcanzada en la última iteración es inferior a la obtenida en la iteración 0 (cuando todavía no se ha realizado entrenamiento semi-supervisado y se cuenta con un *random forest* normal). Sin embargo, sí se puede observar como en general, en las iteraciones centrales, la precisión inicial es mejorada.

Si se observa con más detenimiento cada conjunto de datos individualmente (gráfica 5.3), se puede observar que la desviación típica varía considerablemente, por lo que se puede deducir que el modelo podría llegar a ser utilizable si el conjunto de entrenamiento es el adecuado.

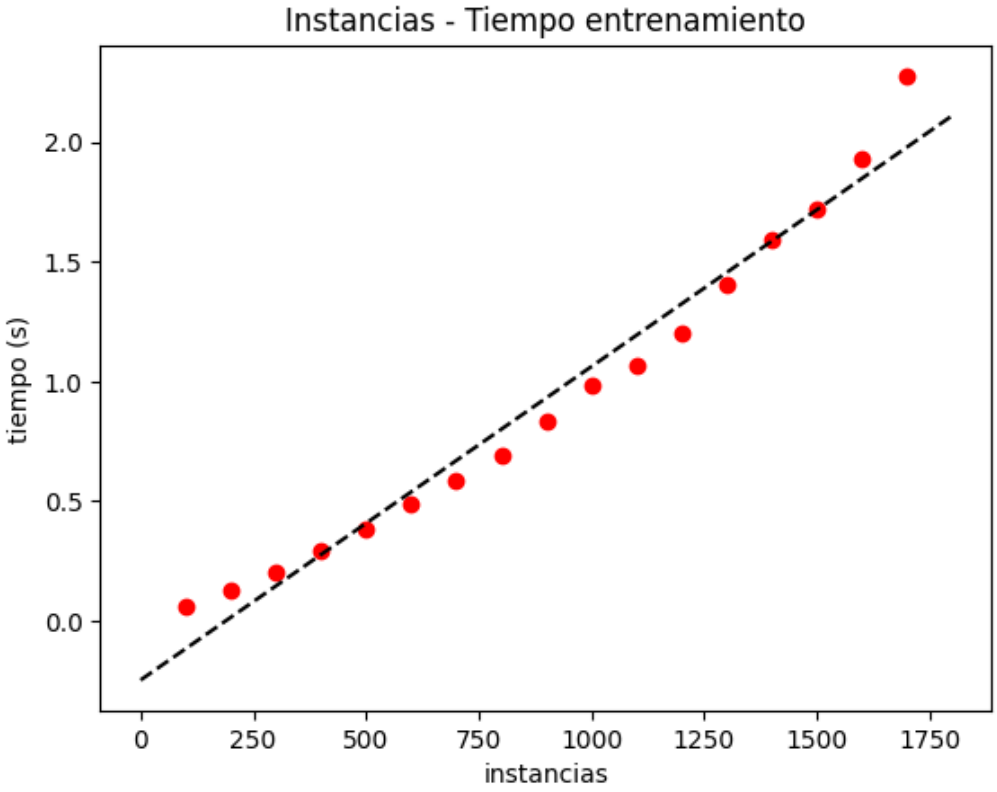
Figura 5.3: Gráfica que representa, para cada conjunto de datos, la precisión media del modelo en función de la iteración en la que se encuentre junto con la desviación típica para 10 experimentos.



Además, también se ha querido evaluar cómo varía el tiempo de entrenamiento en función del número de instancias utilizadas. Para ello, se ha trabajado con el *dataset* que contiene un mayor número de datos, y los resultados obtenidos se pueden observar en la gráfica 5.4. Como se puede

observar, sigue un crecimiento aproximadamente lineal. La velocidad, en este caso, es alta, pero cabe recordar que depende en gran medida del número de árboles utilizados y de las iteraciones que estos realicen (en este caso, 2).

Figura 5.4: Gráfica que representa la evolución del tiempo de entrenamiento en función del número de instancias utilizadas.

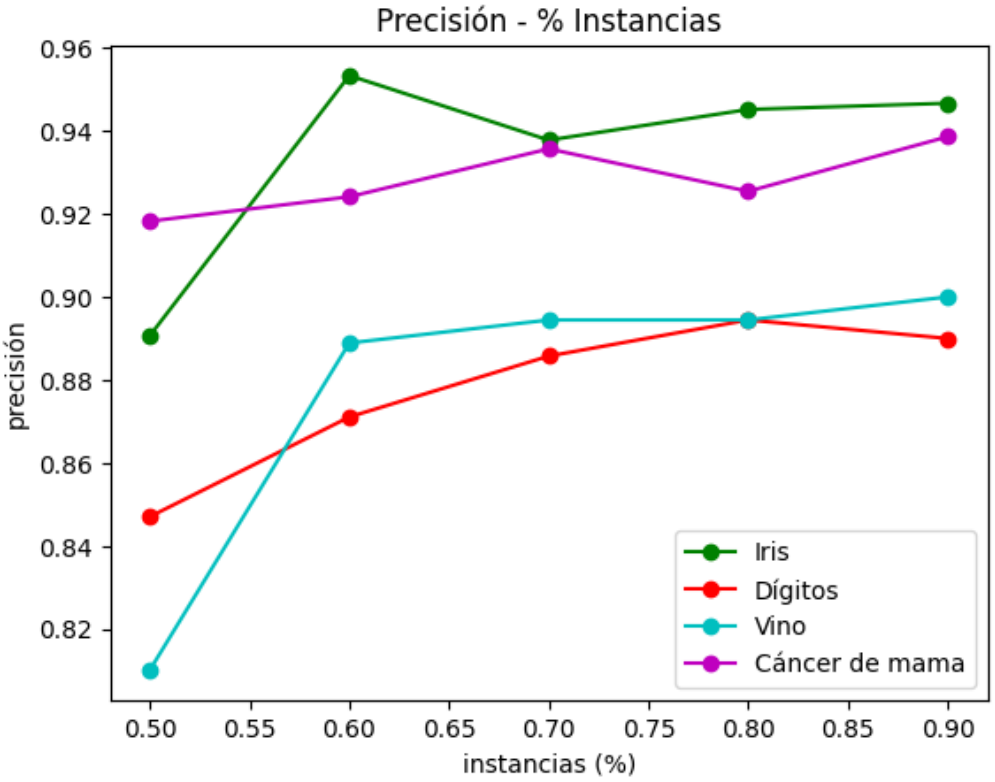


Comportamiento general

En este apartado de la experimentación, se ha querido evaluar cómo se comporta el *co-forest* en función del porcentaje de datos que se utilice para el entrenamiento. Nuevamente, se han utilizado 20 árboles, y la división del conjunto de datos de entrenamiento se ha mantenido: 20 % etiquetados y 80 % no etiquetados.

Como se puede comprobar en la gráfica 5.5, se sigue el comportamiento esperado, y por lo general a más instancias utilizadas para entrenar el modelo, mejor comportamiento desarrolla. Nuevamente, recalcar que el resultado obtenido es la media de 10 experimentos realizados.

Figura 5.5: Gráfica que representa la precisión media del modelo en función del porcentaje de datos utilizados para el entrenamiento.



Se representa además, en la gráfica 5.6 la desviación media obtenida en los 10 experimentos para cada conjunto de datos.

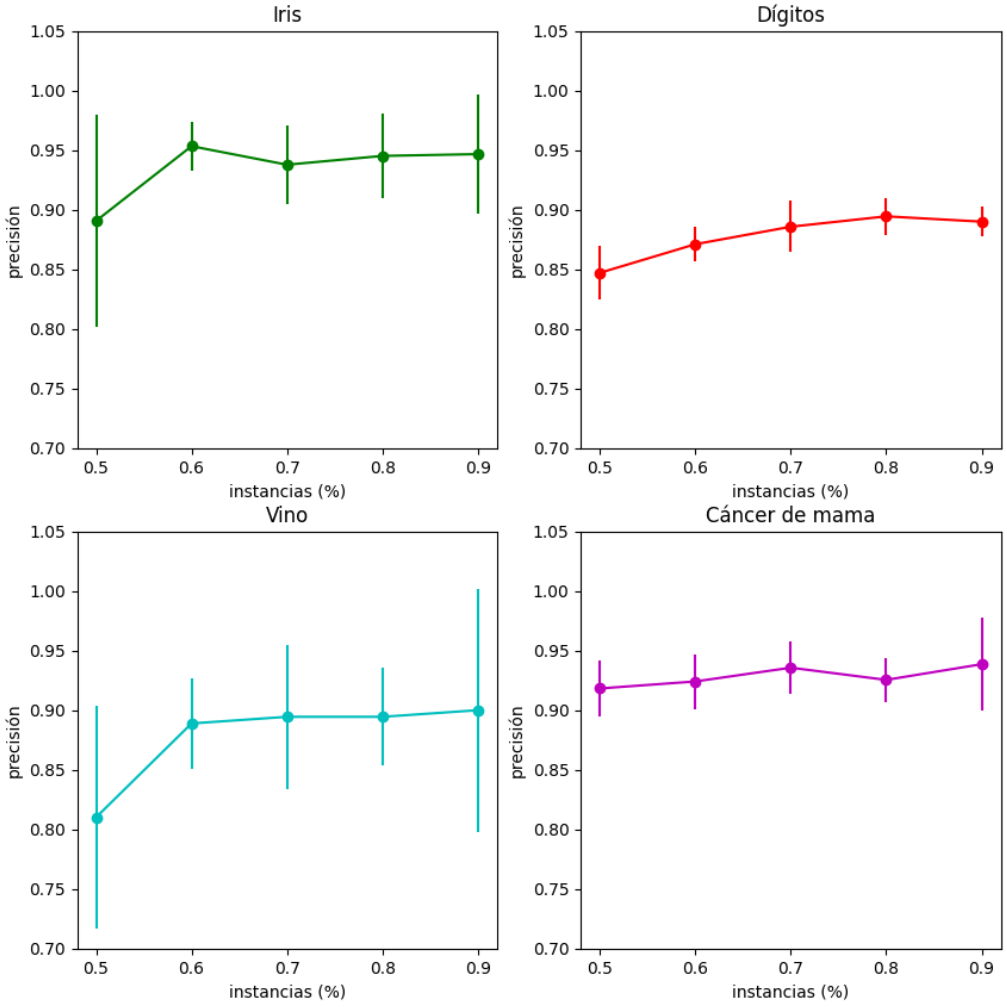
Comparativa contra KEEL

Para comprobar la validez del algoritmo implementado, se ha decidido probar contra la herramienta *KEEL*, creada por distintas universidades españolas y financiada por el Ministerio de Educación y Ciencia.

En primer lugar y para tener más capacidad de «manipulación», se optó por descargar los ficheros fuentes de la última versión de GitHub [3] en lugar de utilizar la versión compilada que ofrecen los desarrolladores.

Una vez se hubo descargado los ficheros fuente, se compilaron aprovechando el fichero *build.xmlz* contenido y la herramienta *ant* mediante los siguientes comandos:

Figura 5.6: Gráfica que representa, para cada conjunto de datos, la precisión media del modelo y su desviación en función del porcentaje del conjunto utilizado para el entrenamiento.




```
# ant cleanAll

# ant
```

.

El primero se utiliza para eliminar binarios previos y evitar conflictos, mientras que el segundo tiene como objetivo compilar el código fuente.

Parámetro	**Valor**
n	6
θ	0.75
Folds	10
% etiquetados en el conjunto de entrenamiento	10 %
Comentarios	Para la comparativa se han utilizado los *datasets* de SKLearn «Iris» y «Vino». El conjunto «Iris» está estratificado (el subconjunto de *test* contiene la misma proporción de clases a probar), mientras que «Vino» no lo está.

Tabla 5.2: Tabla resumen con el diseño del experimento.

Posteriormente se ejecutó la aplicación mediante el comando

```
# java -jar ./dist/GraphInterKeel.jar
```

.

Para comparar los algoritmos en las condiciones más realistas posibles, se definieron las características mostradas en la tabla 5.2.

Además, dentro de los ficheros fuente de KEEL, se pueden encontrar distintos conjuntos de datos con las particiones del *K-cross-validation* ya hechas y que son utilizadas en las ejecuciones de sus experimentos. Para comprobar ambos algoritmos en igualdad de condiciones, se convirtieron dichos archivos **.dat* en **.csv* y se importaron en la implementación propia mediante la librería Pandas. De esta manera, el único parámetro que no es idéntico entre ambas implementaciones es qué instancias de entre los datos etiquetados seleccionan los árboles para entrenarse en un primer momento (son aleatorios y generar las pequeñas diferencias).

Los resultados obtenidos se pueden observar en la tabla5.3.

Como se puede comprobar, ambos algoritmos obtienen resultados prácticamente idénticos, siendo un poco superior el porcentaje de acierto logrado por la herramienta implementada propia.

Fold	**Iris propio**	**Iris KEEL**	**Vino propio**	**Vino KEEL**
Fold 1	0,87	0,87	0,89	0,83
Fold 2	0,86	0,80	0,94	0,94
Fold 3	1,00	1,00	1,00	1,00
Fold 4	1,00	1,00	1,00	0,89
Fold 5	0,93	1,00	0,88	0,88
Fold 6	0,93	0,93	0,76	0,71
Fold 7	0,93	1,00	0,89	0,89
Fold 8	0,93	0,93	1,00	0,78
Fold 9	1,00	0,93	0,89	0,72
Fold 10	0,87	0,87	0,89	0,94
Media	0,93	0,93	0,91	0,86

Tabla 5.3: Comparativa entre la exactitud del *co-forest* de KEEL y el propio sobre el conjunto de *test*.

Trabajos relacionados

Dentro de este proyecto se pueden diferenciar distintas líneas de investigación, principalmente las dirigidas hacia el desarrollo y comprensión de los algoritmos de aprendizaje semisupervisado y las centradas en formalizar ataques a sistemas de recomendación.

Aprendizaje semisupervisado aplicado a la detección de ataques en sistemas de recomendación

Co-Forest aplicado a la detección de ataques [11]

En esta sección, el artículo fundamental es «*Semi-supervised recommendation attack detection based on Co-Forest*» [11]. En este *paper*, se propone un método de detección basado en Co-Forest y se producen distintas comparativas con otros algoritmos para comprobar su eficacia, consiguiendo unos resultados muy aceptables. Partiendo de esta base nace el presente documento, que pretende explorar la solución propuesta por estos autores y expandirla.

Naive Bayes aplicado a la detección de ataques [10]

También es muy relevante citar el trabajo expuesto en «*HySAD: A semi-supervised hybrid shilling attack detector for trustworthy product recommendation*» [10], puesto que propone una aproximación Naive Bayes para separar perfiles de atacantes de perfiles genuinos y además utiliza los tipos de *datasets* que son probados posteriormente por Zhou y Duan (Amazon, Netflix y MovieLens). Se trata de uno de los trabajos de referencia en el área.

Ataques en sistemas de recomendación

La importancia de proteger los sistemas de recomendación ha sido contemplada desde principio de siglo, siendo común la proposición de otros tipos de aprendizaje para detectar los ataques.

Recolección de los tipos de ataques y propuestas de reconocimiento [6]

Respecto a la descripción de los tipos de intrusión, la correcta definición formal (matemática) de sus parámetros y una recopilación de la gran mayoría de ataques existentes, es fundamental referenciar el artículo de Mingdan, Quingshan «*Shilling attacks against collaborative recommender systems: a review*» [6]. Se trata de una recopilación reciente (2018) de las principales investigaciones de los autores más populares en la materia que destaca por su completitud.

Definición de conceptos [9]

Previo a este documento,también es relevante contemplar otros trabajos, como la tesis de William y Mobasher «*Thesis: Profile injection attack detection for securing collaborative recommender systems*» [9]. En ella se introduce el concepto de inyección y se parametrizan características como el tamaño de ataque. Es destacable la autoridad de estos investigadores en la materia, siendo propietarios de muchos documentos de interés.

Primeras definiciones formales [7]

«*Collaborative recommendation: A robustness analysis*» [7], de O'Mahony y Hurley, es uno de los trabajos con más antiguedad pero mayor número de referencias que se encuentra. En él se definen los modelos de construcciones en base a conocimiento del sistema y pone a prueba la robustez de los recomendadores evaluando su estabilidad y precisión ante la presencia de perfiles inyectados (análisis matemático muy completo).

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

Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2:343–370, 1988.

Robin Burke, Michael P. O'Mahony, and Neil J. Hurley. *Robust Collaborative Recommendation*, pages 805–835. Springer US, Boston, MA, 2015.

Jesús Alcalá Fernández (Coordinator). Keel: Github, 2018.

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

Ming Li and Zhi-Hua Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(6):1088–1098, 2007.

Si Mingdan and Qingshan Li. Shilling attacks against collaborative recommender systems: a review. *Artificial Intelligence Review*, 53, 01 2018.

Michael O'Mahony, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol.*, 4(4):344–377, nov 2004.

Jesper Van Engelen and Holger Hoos. Semi-supervised ensemble learning. master's thesis., 07 2018.

Chad Williams, Research Advisor, and Bamshad Mobasher. Thesis: Profile injection attack detection for securing collaborative recommender systems, 2006.

Zhiang Wu, Junjie Wu, Jie Cao, and Dacheng Tao. Hysad: A semi-supervised hybrid shilling attack detector for trustworthy product recommendation. page 985–993, 2012.

Quanqiang Zhou and Liangliang Duan. Semi-supervised recommendation attack detection based on co-forest. *Comput. Secur.*, 109(C), oct 2021.