UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Aprendizaje semisupervisado y ciberseguridad: detección automática de ataques en sistemas de recomendación y *phishing* Documentación técnica.

Presentado por Patricia Hernando Fernández
en Universidad de Burgos — 6 de abril de 2023
Tutor: Álvar Arnaiz González

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este documento se pretende mostrar el plan de proyecto seguido a la hora de realizar el trabajo de fin de grado.

Debido al auge de las metodologías ágiles en la industria y sus indiscutibles ventajas, se ha escogido la metodología Scrum para desarrollar el proyecto. Los conceptos teóricos seguidos se encuentran en el «Manual de Scrum» [6], temario necesario para obtener la certificación de Scrum Manager.

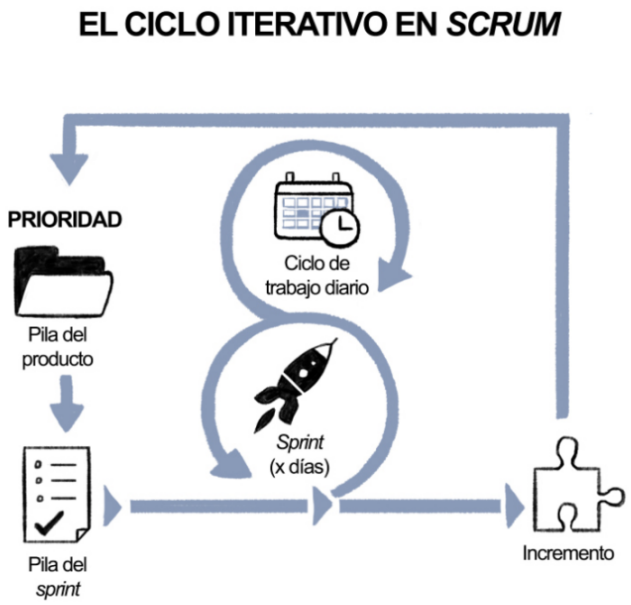
Posteriormente, se expondrá la viabilidad legal y económica del producto.

A.2. Scrum

Scrum es una metodología ágil basada en cuatro valores [6]. Sintetizando sus principios, se puede deducir que se pretende valorar a los individuos por encima de las herramientas, el *software* apropiado a la documentación exhaustiva, la colaboración con el cliente y la habilidad de dar respuesta al cambio ante imprevistos.

Siguiendo este esquema, se desarrolla el «ciclo de Scrum» representado en la imagen A.1, que se puede dividir en varios pilares.

Figura A.1: Resumen del ciclo de Scrum según el Manual de Scrum [6].



Roles

Descripción de los posibles interventores durante el desarrollo de un producto.

- *Product owner*: es el representante del cliente, y su responsabilidad es el valor del producto. Es quien gestiona la pila del producto (conjunto de historias de usuario solicitadas por el cliente), así como la prioridad de cada ítem que lo compone.
- Scrum *master*: es el responsable de garantizar que el proyecto se desarrolla siguiendo los principios de Scrum, asesorando a los desarrolladores. También es el moderador en las reuniones diarias de Scrum y el encargado de resolver dinámicas que puedan perjudicar al equipo.
- Equipo de desarrollo: es el conjunto de programadores autogestionados encargados de generar los incrementos. Profesionales multifuncionales, deben completar las tareas que se les asignen en el plazo estimado, además de participar en la toma de decisiones.

Artefactos

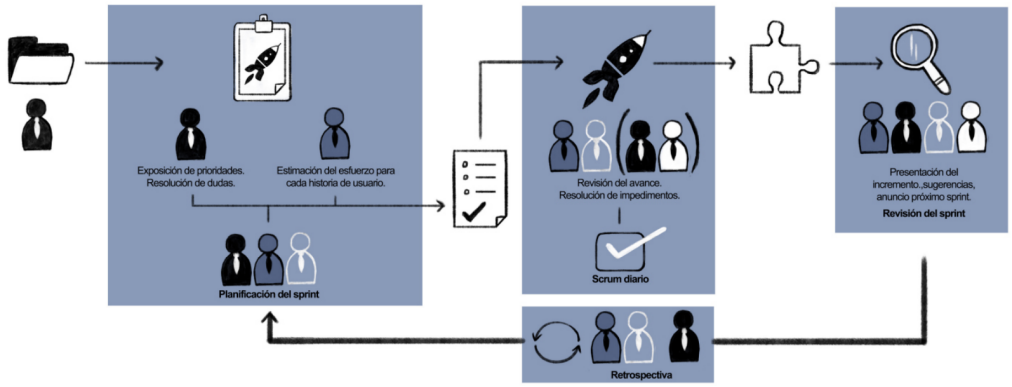
Son las «herramientas» [6] elementales. Entre ellos se encuentran:

- Pila de producto o *product backlog*: contiene las historias de usuario (el equivalente a los «requisitos» en las metodologías tradicionales). Evoluciona durante el proyecto en función de las peticiones del cliente.
- Pila del *sprint* o *sprint backlog*: es la lista de tareas que han de completar los desarrolladores en un *sprint*. En este proyecto, se han utilizado las *milestones* de GitHub (asignándoles *issues*) para representarla.
- Incremento: resultado de cada *sprint*. Ha de ser un entregable.
- Gráfico de avance o *burn down report*: muestra el trabajo pendiente por realizar en un *sprint* y es actualizado por los desarrolladores. Indica, además, el ritmo de trabajo «ideal» que se debería seguir para alcanzar los objetivos. En este caso, se ha utilizado el gráfico generado por ZenHub.
- Gráfico de producto o *burn up report*: mide cuánto se ha completado respecto al total.

Eventos

Durante el ciclo de Scrum, se pueden identificar varias actividades que constituyen la rutina y se represetan en la imagen A.2 facilitada por el Manual de Scrum [6].

Figura A.2: Eventos de Scrum según el Manual de Scrum [6].



1. *Sprint* o iteración: es un periodo de tiempo fijo (y breve) en el que se trabaja para completar una cantidad de tareas prefijadas con antelación (la pila del *sprint*). Se han utilizado los *sprints* de ZenHub para realizar el seguimiento.

2. Reunión de planificación sel *sprint*: marca el inicio de cada *sprint* y determina las tareas a desarrollar, además de su duración temporal.

3. Scrum diario: breve reunión en la que cada integrante del equipo notifica su ritmo de trabajo con el fin de corregir posibles impedimentos que ralenticen el ciclo. Además, se actualiza el *burndown report*. En este caso, debido a que el equipo está formado únicamente por un desarrollador, se ha omitido.

4. Revisión del *sprint*: se analiza el incremento entregado y se adapta la pila del producto en caso de necesitarlo (por ejemplo, si se ha encontrado algún *bug* o se necesita refactorizar).

5. Retrospectiva del *sprint*: se aporta el *feedback* necesario para mejorar la siguiente iteración.

Medición

Como se puede observar, cada equipo puede realizar una cantidad de trabajo, generalmente fija, en cada *sprint*. Por ello, es necesario estimar el tiempo que requiere cada tarea y no asignar más trabajo del que se pueda asimilar en cada iteración.

En este proyecto, se han utilizado los «puntos de historia» como unidad de medida.

A.3. Planificación temporal

Planificación por *sprints*

Siguiendo los eventos de Scrum, y adaptándolos teniendo en cuenta el tamaño del equipo (un desarrollador), se ha decidido planificar el proyecto mediante *sprints*.

Sprint 1: Mustard

- ***Planning meeting***

  Durante la reunión se marcaron los siguientes objetivos:

  1. Configuración básica: incluyendo la creación del repositorio, la correcta instalación de ZenHub, la creación de entornos virtuales

(miniconda, Scikit-Learn, etc.) y la familiarización con conceptos Scrum: *milestones, sprints, epics*, etc.

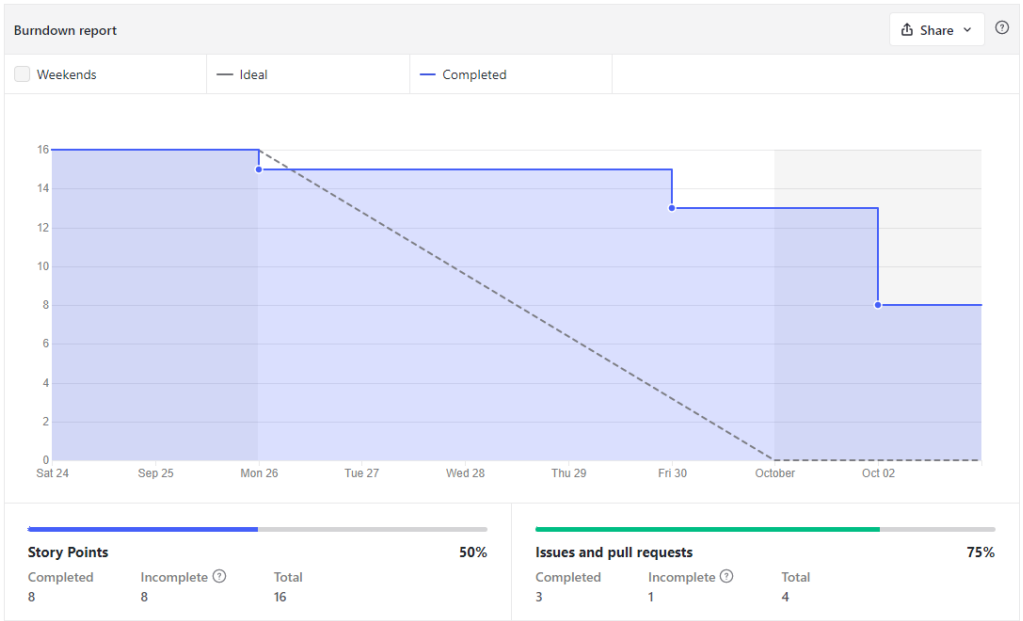
2. Memoria: comienzo de la redacción incluyendo las secciones de introducción, conceptos teóricos (aprendizaje automático) y trabajo relacionado.
3. Investigación: búsqueda del código SSADR-CoF y de las bases de datos utilizadas en el paper.
4. Lectura de papers: Engelen & Hoos [9], García, Triguero & Herrera [7], y Zhou & Duan [10].

- **Marcas temporales**

  El *sprint* se desarrolló entre el 24 de septiembre de 2022 y el 2 de octubre del 2022.

- ***Burndown Report***

Figura A.3: *Burndown Report Sprint 01*



Como se puede comprobar, no todos los objetivos marcados fueron cumplidos: la estimación del tiempo fue demasiado optimista, además de no contar con el tiempo requerido en solucionar problemas técnicos (LaTeX). Se dejó para próximos sprints la lectura del último paper.

- ***Sprint review meeting***

  Durante la reunión se fijaron ciertas correcciones en la memoria (mejorar referencias bibliográficas y la sección de «Trabajos relacionados»), además de la necesidad de introducir una sección teórica de ataques a los sistemas de recomendación.

Sprint 2: Paprika

- ***Planning meeting*** Objetivos del siguiente Sprint:

  1. Configuración: debido a la gran cantidad de tiempo invertida en solucionar errores de compilación en LaTeX, se decidió migrar el proyecto a una nueva instalación basada en Debian.
  2. Correcciones: aspectos estilísticos y completar información.
  3. Lectura: Mingdan y Qingshan [5] con el objetivo de introducir una sección teórica de ataques.
  4. Memoria: redacción completa de los modelos de ataque en los aspectos teóricos.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 3 de octubre de 2022 y el 18 de octubre del 2022.

- ***Burndown Report***

  En este *sprint* sí se cumplió con los objetivos marcados. Sin embargo, la estimación de tiempo tampoco fue la adecuada, requiriendo más de lo previsto.

- ***Sprint review meeting***

  Durante la reunión se resolvieron dudas acerca de bibliografía, referencias y trabajo previo. Además, se acordó empezar a programar, definiendo así los *issues* desarrollados en el siguiente *sprint*.

Sprint 3: Fennel Seeds

- ***Planning meeting***

  Durante esta reunión, se decidió empezar a programar el *co-forest*. Para ello, se definieron los siguientes pasos:

Figura A.4: *Burndown Report Sprint 02*

1. Librerías: se acordó aprender a utilizar las librerías más comunes en el *data science*. Entre ellas: MatplotLib y Scikit-Learn. Además, se requirió la correcta configuración del entorno virtual, haciendo que el tiempo dedicado al *issue* fuese mayor de lo estimado (problemas en el `PATH` y con las dependencias).
2. *SKLearn*: aprovechando la correcta documentación de la librería, se decidió repasar los conceptos teóricos básicos, además del manejo de la «interfaz» (métodos comunes). Entre ellos:
   - *Decision trees*
   - *Self training*
   - *Random Forest*
3. Lectura: se concertó la relectura del artículo de Zhou [10] con la intención de comprender el algoritmo y del *paper* «original» del *co-forest* [4]. Durante el proceso de programación, además, se encontró la tesis de Van Engelen [8] y se añadió al conjunto.
4. Documentación: se acordó la corrección de los errores previamente señalados y la inclusión del *sprint* en los anexos.
5. Programación del *co-forest*: se programó el pseudocódigo ilustrado en la Tesis de Van Engelen [8], que es muy similar al original [4]

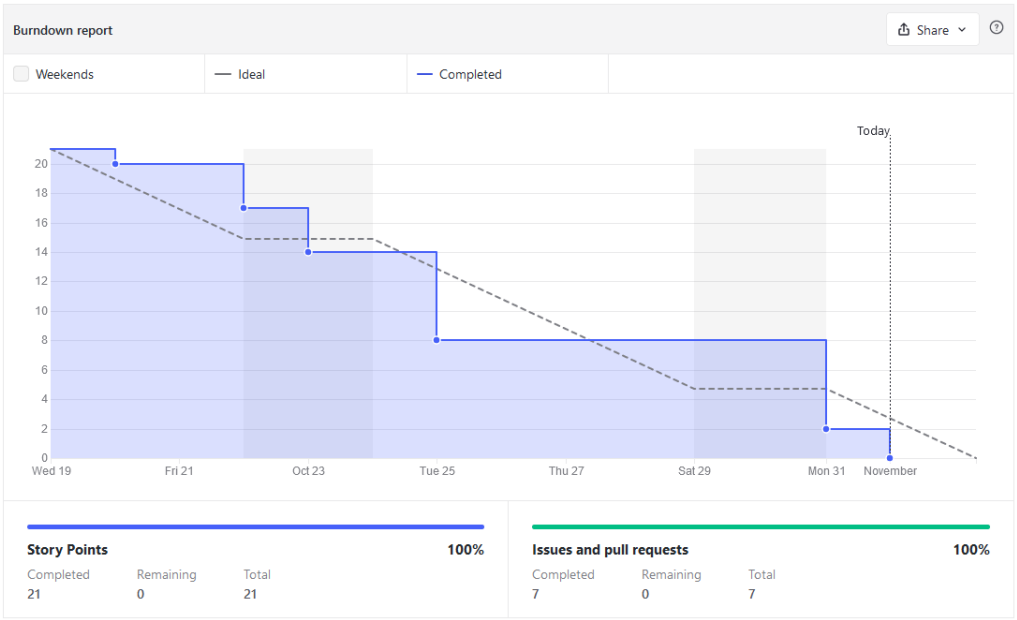
pero con algunas diferencias. Inicialmente se intentó usar el *random forest* de Scikit-Learn, pero se descartó la idea debido a la poca versatilidad que se ofrecía para manejar los *concomitant ensembles*. Se han de corregir ciertos factores, pero se pospondrá hasta la correcta discusión con el tutor.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 19 de octubre de 2022 y el 2 de noviembre del 2022.

- ***Burndown Report***

Figura A.5: *Burndown Report Sprint 03*



En este *sprint* se cumplió con los objetivos marcados. Nuevamente, la estimación del tiempo fue inferior a la real (se pensaba que se podría depender más de librerías existentes de lo que se pudo en realidad), calculándose un total de aproximadamente 25 horas reales.

- ***Sprint review meeting***

  Durante la revisión del *sprint*, se llegó a la conclusión de que el pseudocódigo podía ser mejor implementado aprovechando ciertas librerías de Python. Se comentó cómo mejorar complejidades espaciales y reducir el código. Se fijaron objetivos para las próximas semanas.

Sprint 4: Cayenne

- ***Planning meeting***

  Durante la reunión se acordaron los siguientes objetivos:

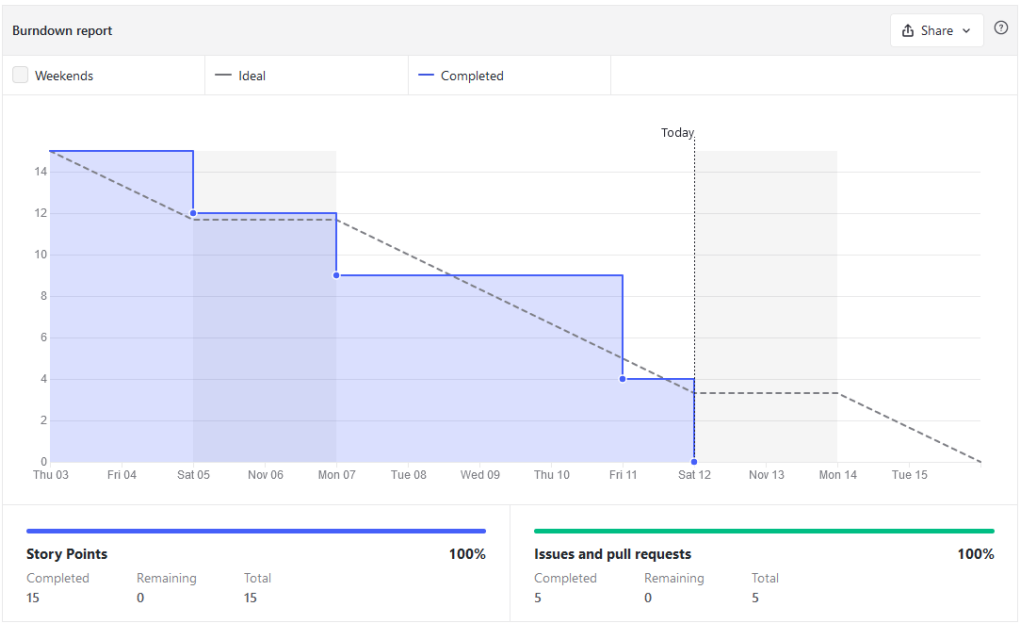
  1. Reimplementación del código: se acordó volver a programar el *co-forest*, esta vez implementando una versión más «*pythonizada*» con el fin de mejorar la complejidad espacial y facilitar la lectura.
  2. Curso de Numpy: se decidió que sería interesante la realización de un curso para aprender a utilizar la librería y aplicarla al código.
  3. Curso de Pandas: aprovechando la relación con el punto anterior, se acordó completar también un curso de esta librería.
  4. Memoria: corregir aspectos anteriores e incluir toda la teoría relacionada con el *co-forest*.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 3 de noviembre de 2022 y el 15 de noviembre del 2022.

- ***Burndown Report***

Figura A.6: *Burndown Report Sprint 04*



En este *sprint* se completaron los objetivos, aunque quedaron pendientes ciertos aspectos a comentar respecto al código. Destacar que, debido a que se terminaron antes de lo planeado los *issues* planificados, se aprovechó para modificar ciertos aspectos pendientes relacionados con la memoria y para probar correctamente el código. Esto hizo que el tiempo real dedicado haya sido ligeramente superior al estimado (más tiempo de documentación).

- ***Sprint review meeting***

  En la reunión se acordó experimentar con el algoritmo utilizando distintos conjuntos de datos, además de mejorar ciertos detalles de implementación.

Sprint 5: Curry

- ***Planning meeting***

  Durante la reunión se acordaron los siguientes objetivos:

  1. Ajustes al código: se acordó mejorar algunos factores, como la generación de objetos «aleatorios» para obtener resultados deterministas en los experimentos, optimización de memoria o corrección de parámetros.
  2. Experimentación: se determinó probar el código con distintos conjuntos de datos en diferentes fases: durante el entrenamiento y tras terminarlo. Para ello, se estudiaron algunos conceptos teóricos y el uso de la librería MatplotLib para representar gráficamente los resultados obtenidos.
  3. Memoria: documentación de los experimentos realizados y correcciones.

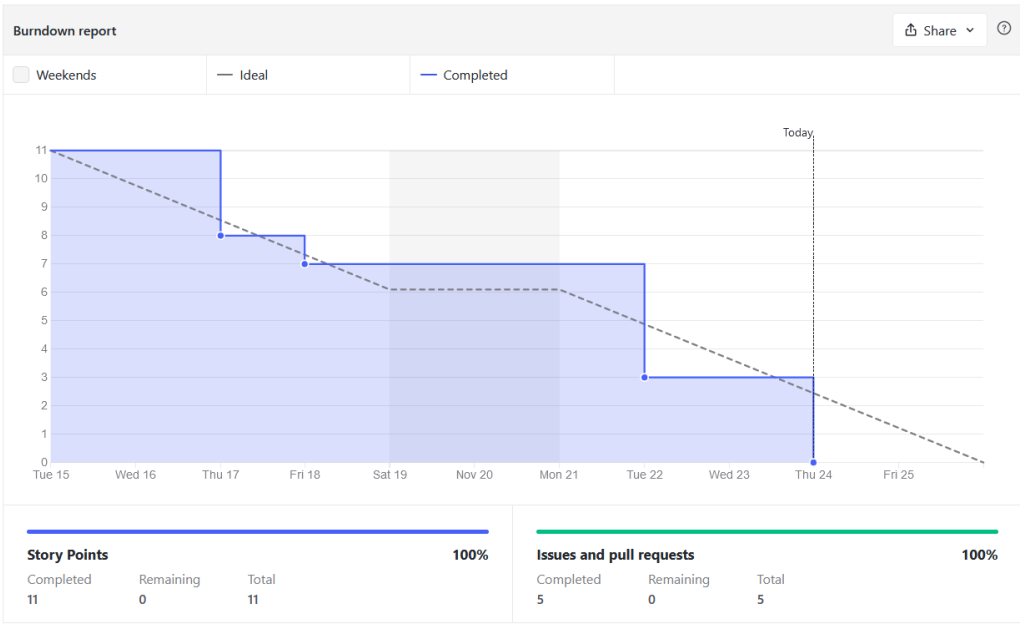
- **Marcas temporales**

  El *sprint* se desarrolló entre el 15 de noviembre de 2022 y el 25 de noviembre del 2022.

- ***Burndown Report***

  En este *sprint* se cerraron todos los puntos de historia propuestos. Aunque se estimaron 11 horas de trabajo, el tiempo invertido fue superior, realizando 15. La mayor dedicación se justifica por la existencia de reuniones intermedias y de «defectos» encontrados en el código en el trascurso del *sprint*.

Figura A.7: *Burndown Report Sprint 05*



- ***Sprint review meeting***

  Durante la reunión se acordó comparar los resultados obtenidos con los de otras herramientas, además de empezar el tratamiento de los conjuntos de datos utilizados en el *paper* [10].

Sprint 6: Coriander

- ***Planning meeting***

  Durante la reunión se fijaron los siguientes objetivos:

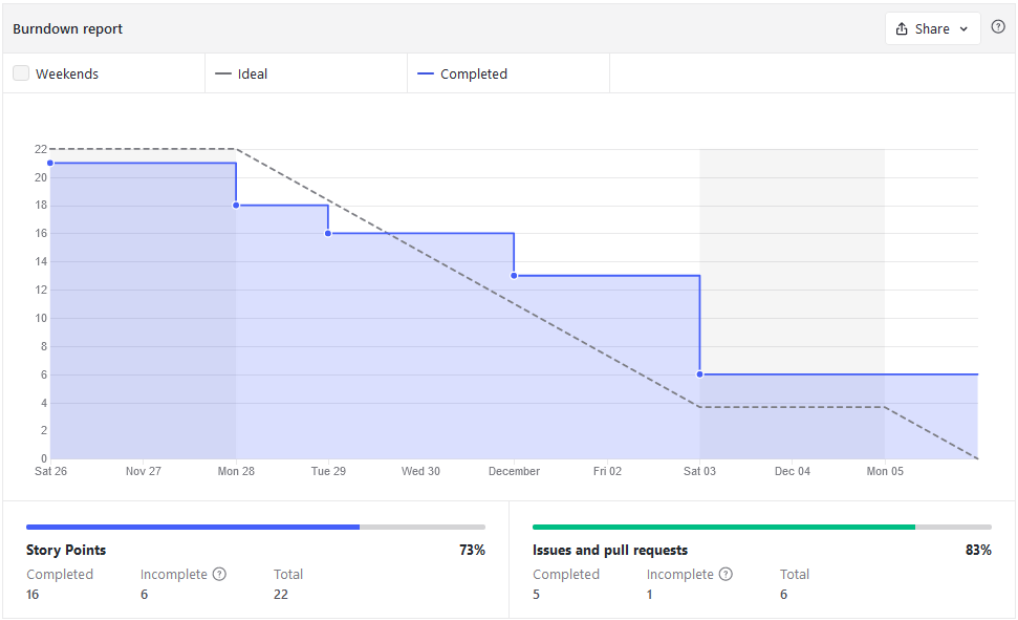
  1. Ajustes en las gráficas: arreglar detalles menores en el formato de ciertos gráficos.
  2. Comparativas: probar los resultados obtenidos y compararlos la herramienta de la Universidad de Granada llamada KEEL.
  3. *Datasets* «reales»: probar el algoritmo utilizando MovieLens, una de las bases de datos utilizadas en el *paper* [10]. Para ello, es necesaria una re-lectura del artículo y realizar el procesamiento inicial de los datos.
  4. Memoria: documentación de los experimentos realizados.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 25 de noviembre de 2022 y el 5 de diciembre del 2022.

- ***Burndown Report***

Figura A.8: *Burndown Report Sprint 06*



Puede parecer que los puntos de historia fueron mal estimados debido a que el ritmo de trabajo es bajo. Sin embargo, se justifica debido a que durante la experimentación (en concreto, durante la comparativa contra KEEL) se encontraron dos *bugs* en el código (causados no por la lógica del programa, sino por el operador *in* de Python y por no estar preparado para recibir etiquetas que no comiencen en 0).

Debido a que los errores se encontraron una vez se realizó toda la documentación, se tuvo que repetir la sección asociada a los experimentos del *co-forest*, además de localizar y depurar los errores de código. Todo este trabajo supuso un esfuerzo extra de 7 horas, que fueron introducidas en el *backlog* del *sprint* a mediados del mismo (debido a la gran importancia que tienen y la influencia en pasos posteriores). Por este motivo, no se completaron los objetivos previstos. Sin embargo, el tiempo dedicado al proyecto fue el estimado.

Es destacable también que de los 6 puntos de historia que quedan se realizaron 2, pero se decidió dejar el *issue* abierto para el siguiente *sprint*.

- ***Sprint review meeting***

  Habiendo finalizado el modelo, se decidió empezar a experimentar con bases de datos de sistemas de recomendación. Además, se acordó añadir nuevas gráficas.

Sprint 7: Cinnamon

- ***Planning meeting***

  Se marcaron los siguientes objetivos para el *sprint*.

  1. Extraer vectores de características: estudiar e implementar el método de extracción de vectores por ventanas expuesto en el *paper* de Zhou y Duan [10]. Generar ficheros `.csv` con dichos vectores para poder importarlos posteriormente con Pandas. Probar la correcta generación de vectores con perfiles verdaderos y atacantes.
  2. Generación de reseñas de atacantes: siguiendo los modelos estadísticos, generar reseñas de ataque para el *random attack*, el *average attack* y el *bandwagon attack*.
  3. Documentación: añadir introducción y descripción de Scrum en los anexos. Corregir las sugerencias anteriores. Incluir la descripción del método de extracción de vectores de características por ventanas y la generación de reseñas de atacantes.

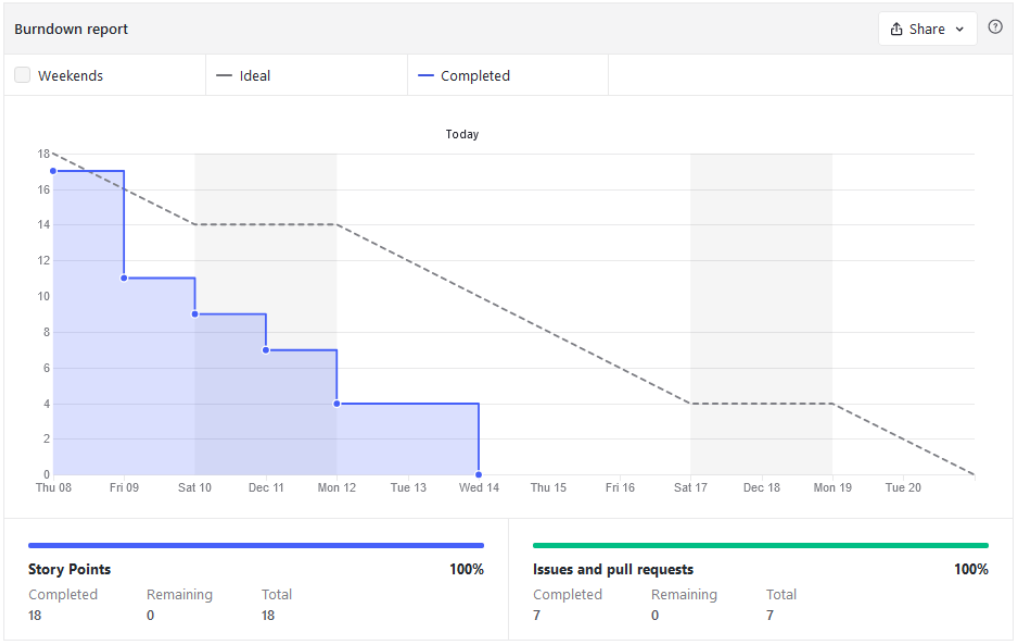
- **Marcas temporales**

  El *sprint* se planificó para desarrollarse entre el 8 y el 20 de diciembre de 2022. Sin embargo y debido al ritmo de trabajo, se cerró el 15 de diciembre de 2022.

- ***Burndown Report***

  Como se puede comprobar, se finalizaron las tareas del *sprint* antes de lo previsto. En gran parte, debido a que el *issue* de extracción de vectores de características estaba ya comenzado en el *sprint* anterior, y requirió menos tiempo del planificado. Además, el resto de tareas no dieron problemas en esta iteración. Debido a que Scrum no permite

Figura A.9: *Burndown Report Sprint 07*



añadir nuevos ítems en la pila del *sprint* durante el desarrollo de este, se decidió cerrar y comenzar uno nuevo.

- ***Sprint review meeting***

  Durante la reunión se concluyó que la forma de generar vectores de características y reseñas de ataques es correcta, por lo que se decidió experimentar con el *co-forest* y datos reales.

Sprint 8: Kaffir Lime Leaves

- ***Planning meeting***

  Durante este *sprint* se fijaron los siguientes objetivos:

  1. Refactorizar el extractor de vectores de características y el generador de reseñas de atacantes: debido a que la sintetización de los conjuntos de entrenamiento y *test* requiere demasiado tiempo ($25h$), se ha decidido mejorar la complejidad algorítmica y convertir a clase con el fin de reducir llamadas con mayor complejidad temporal. Se redujo el tiempo requerido a ($10h$) mediante el uso de la nueva estructura (en lugar de fichero de utilidades).

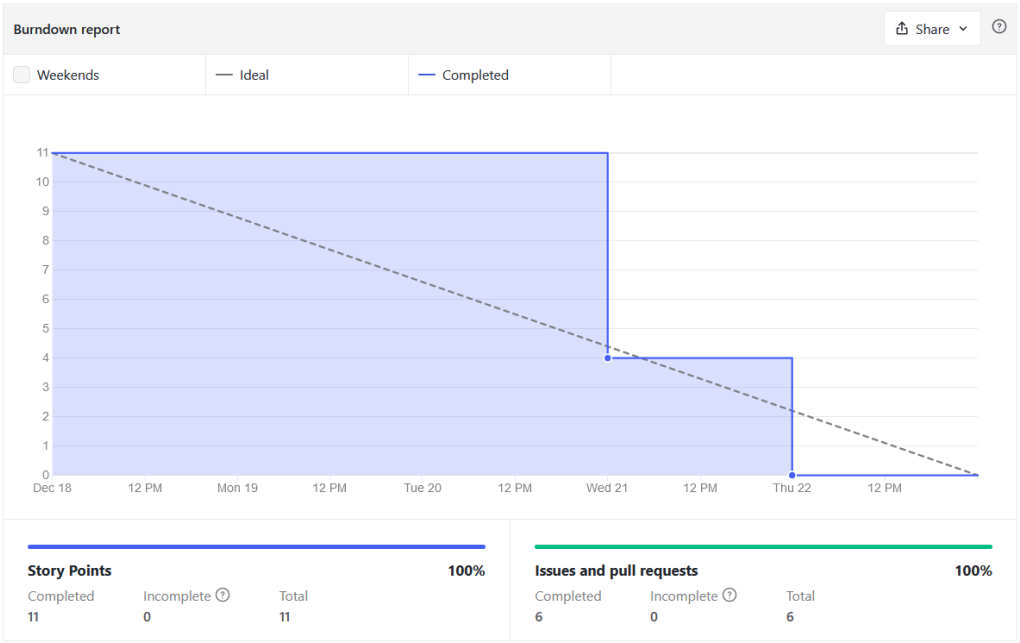
2. Completar el *co-forest*: añadir nuevos métodos para realizar comparaciones (predicción con probabilidades, AUC, *recall* y precisión).
3. Orden de repositorio: lectura acerca de la creación de módulos en python, conversión de los *notebooks* a archivos `.py` importables, configuración de rutas.
4. Análisis de resultados: analizar los resultados en busca de posibles errores cometidos durante la experimentación.
5. Generación de gráficas y comparación contra *random forest*: utilizando clases de Scikit-Learn, automatizar la generación de gráficas para imitar el experimento realizado por Zhou y Duan [10] para el *co-forest* y dos *random forest* con distintos conjuntos de entrenamiento.

- **Marcas temporales**

  Para evitar solapamientos con el *sprint* anterior, oficialmente (en el repositorio) se desarrolló entre el 18 y el 22 de diciembre del 2022. Sin embargo, se empezó unos días antes.

- ***Burndown Report***

Figura A.10: *Burndown Report Sprint 08*



Como se puede comprobar, los *issues* fueron completados correctamente.

- ***Sprint review meeting***

  Durante la reunión se comprobó que los resultados obtenidos en la fase de experimentación no se asemejan a los presentados en el *paper* seguido [10]. Sin embargo, se concluyó que podrían ser correctos. Para evitar realizar afirmaciones erróneas, se decidió revisar todo el proceso de experimentación.

Sprint 9: Ginger

- ***Planning meeting***

  Debido a que este *sprint* se realizó durante la época vacacional, se decidió realizar una revisión general del documento completando aspectos pendientes.

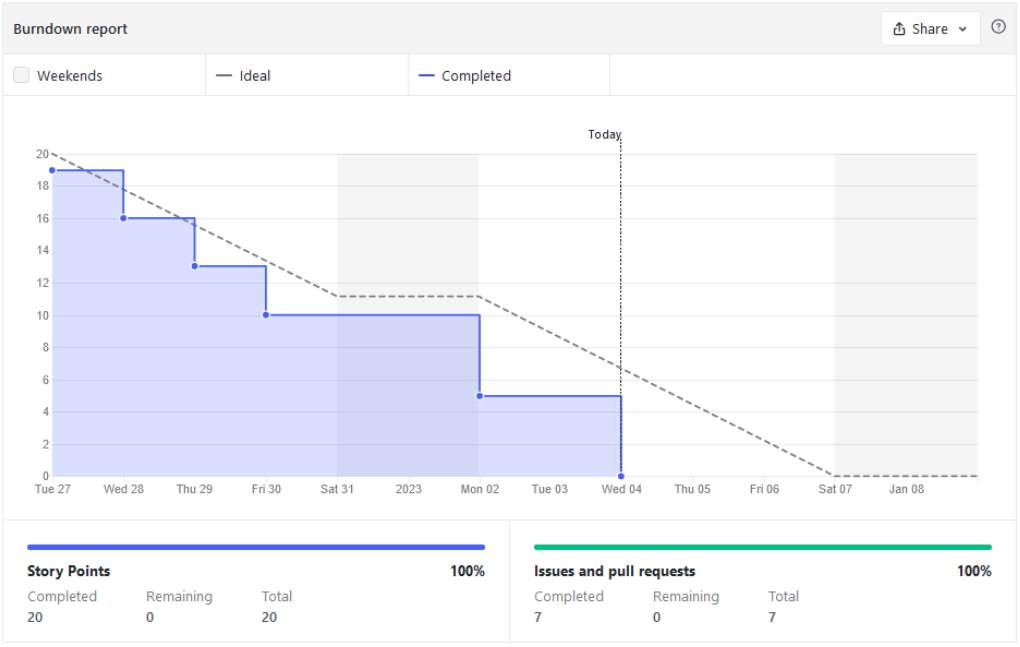
  1. Experimentación *co-forest*: se incluyó una gráfica *extra* pendiente (tiempo de entrenamiento-número de árboles) y se repitió la experimentación esta vez utilizando conjuntos estratificados. Se corrigieron los cambios surgidos y actualizaron las gráficas en la memoria. Se revisó la clase del *co-forest* y se terminó de documentar, además de adaptar los métodos a la interfaz de SKLearn (cambio de firma).
  2. Documentación: se decidió cerrar aspectos pendientes. Entre ellos: completar las secciones teóricas de semisupervisado, ensembles y métricas, repetir gráficas e ilustraciones incorrectas, revisar la bibliografía y trabajos relacionados, completar los anexos y corregir aspectos menores.
  3. Revisión de la detección de ataques: debido a que los resultados obtenidos no son los esperados, se decidió revisar toda la generación de reseñas de atacantes, extracción de perfiles y métodos de experimentación en busca de posibles errores. Se incluyeron alternativas para la AUC (curva *precision* - *recall*).
  4. Implementación: se investigó acerca del *tri-training* y se implementó este nuevo método.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 27 de diciembre del 2022 y el 8 de enero del 2023.

- ***Burndown Report***

Figura A.11: *Burndown Report Sprint 09*



El número total de horas dedicadas al proyecto fue de 20. Aunque se «dejó abierto» el *sprint* hasta el 8 de enero, la intención era acabarlo el 4 de enero, por lo que la estimación temporal fue correcta.

- ***Sprint review meeting***

  Tras haber finalizado la experimentación, se decidió investigar nuevos temas para dedicar el tiempo restante a otros con mejor método de extracción de vectores de características.

Sprint 10: Amchur

- ***Planning meeting***

  En este *sprint* se decidió:

  1. Lectura de aplicación del semisupervisado a la intrusión de redes: lectura del paper de BigFlow y su mejora semisupervisada. Investigación de la base de datos.
  2. Búsqueda de aplicaciones del semisupervisado a nuevos temas: entre ellos NIDS, *phishing*, detección de *malware*, de *markets* en

la *deep web*, de intrusiones en redes IoT, etc. Búsqueda de bases de datos.

3. Continuación de implementación del *democratic co-learning*: implementación del algoritmo y lectura del paper.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 9 y el 20 de enero. Fueron dedicadas al *sprint* 9 horas de trabajo. No se incluye un *burndown report* debido a que no se asociaron las horas de investigación a ningún repositorio en concreto.

- ***Sprint review meeting***

  Tras haber comentado las distintas ramas de investigación posibles, en esta reunión se decidió cambiar el tema principal a la aplicación del aprendizaje semisupervisado para la detección de *phishing*.

Sprint 11: Parsley

- ***Planning meeting***

  Durante esta reunión se acordó comenzar a trabajar en el nuevo objetivo, además de completar documentación pendiente.

  1. Extracción de vectores de características de URLs: generar el código que permita diferenciar un enlace de *phishing* de uno verídico.
  2. Pruebas unitarias: debido a que se ha creado una clase de utilidades para apoyar la extracción de los vectores de características, se ha decidido investigar acerca de las pruebas unitarias en Python e implementar algunas.
  3. Creación de instancias de Tor: debido a que algunos de los métodos de los vectores de características requieren hacer peticiones a páginas peligrosas, se ha decidido proteger esas peticiones utilizando instancias de Tor, de manera que se oculte la IP del ordenador que realiza las peticiones. Para ello, se ha creado un *script* capaz de levantar estas conexiones y se han utilizado como *proxies* (protocolo SOCKS5).
  4. Documentación y experimentación: se ha generado la documentación relativa al algoritmo *tri-training*, además de su comparativa y estudio (gráficas).

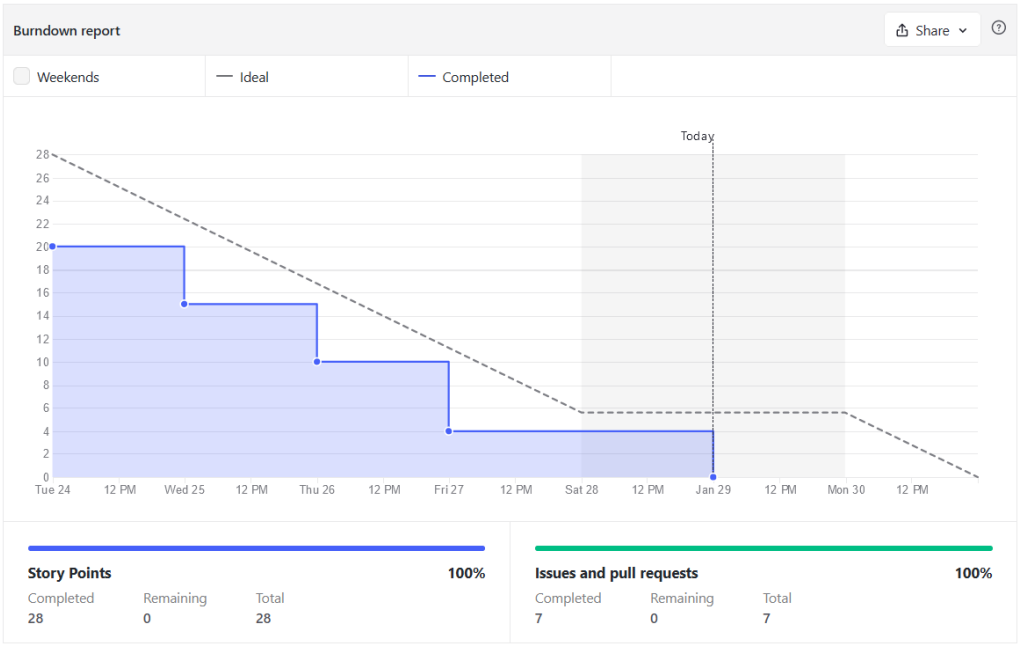
5. Refactorización: debido a que muchos de los métodos para realizar las gráficas del *tri-training* duplican código de los utilizados para generar el *co-forest*, se ha decidido realizar métodos comunes y una clase de utilidades para las gráficas.

- **Marcas temporales**

  El *sprint* fue completado entre el 24 y el 30 de enero del 2022.

- ***Burndown Report***

Figura A.12: *Burndown Report Sprint 11*



El número total de horas dedicadas al proyecto fue de 30, requiriendo una cifra ligeramente inferior a la estimada. Aún así se cerró antes de tiempo debido a que se pudieron invertir más horas por día de las predichas. Nuevamente, recalcar que el tiempo sobrante en ciertos *issues* fue destinado a la generación de vectores de características.

- ***Sprint review meeting***

  En la reunión se concluyó que el trabajo desarrollado es correcto, a excepción de un detalle a cambiar respecto al algoritmo TF-IDF.

Sprint 12: Oregano

- ***Planning meeting***

  En este *sprint* se decidió, principalmente, avanzar la documentación. Por lo tanto, se acordó:

  1. Documentar los resultados de la aplicación de *machine learning* a la detección de ataques en sistemas de recomendación.
  2. Documentar la parte teórica de ciberseguridad, protocolos (SOCKS5), Tor, etc., además de la creación del *script* para levantar instancias.
  3. Documentar y realizar la comparativa del *tri-training*.
  4. Documentar algunas de las herramientas utilizadas hasta ahora.
  5. Documentar la generación de vectores de características para la detección de phishing, además de revisar el código.

  Adicionalmente se solucionaron algunos aspectos no relacionados con la documentación:

  1. Arreglar el uso del algoritmo TF-IDF e integrarlo en la generación de vectores para la detección de *phishing*.
  2. Creación de métodos que recopilen las URLs de las que se extraerán los vectores de características, además de depurar algunos de los `csv` que se necesitan para los enlaces persistentes.

- **Marcas temporales**

  El *sprint* se desarrolló entre el 31 de enero y el 6 de febrero del 2023.

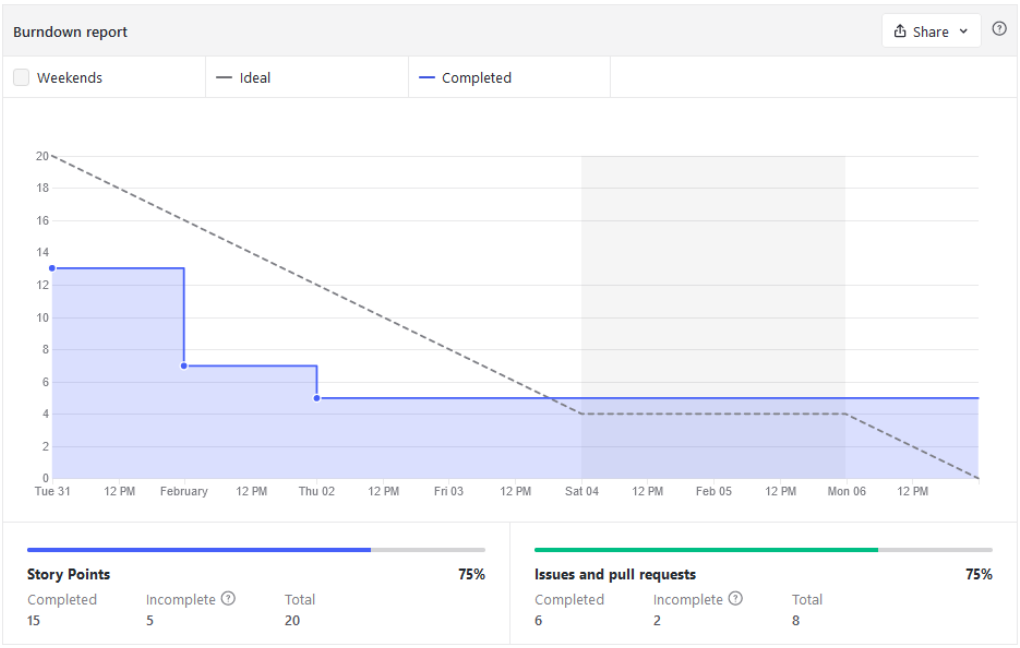
- ***Burndown Report***

  En este *sprint*, no se pudieron cerrar todos los *issues* planeados debido a que algunos de los completados requirieron más tiempo de lo estimado (por ejemplo, los métodos de obtención de URLs debido a que un servidor bloqueó la IP del equipo por exceso de peticiones o la comparativa del *tri-training* debido a que se decidió cambiar la estética de las gráficas de toda la memoria).

- ***Sprint review meeting***

  Tras haber resuelto los problemas que surgieron en el *sprint* y revisado la documentación, se decidió continuar con el proyecto.

Figura A.13: *Burndown Report Sprint 12*



Sprint 13: Tandoori Masala

- ***Planning meeting***

  En este *sprint* se decidió continuar con el algoritmo *democratic-co* learning, además de seguir documentando e iniciar el control de calidad.

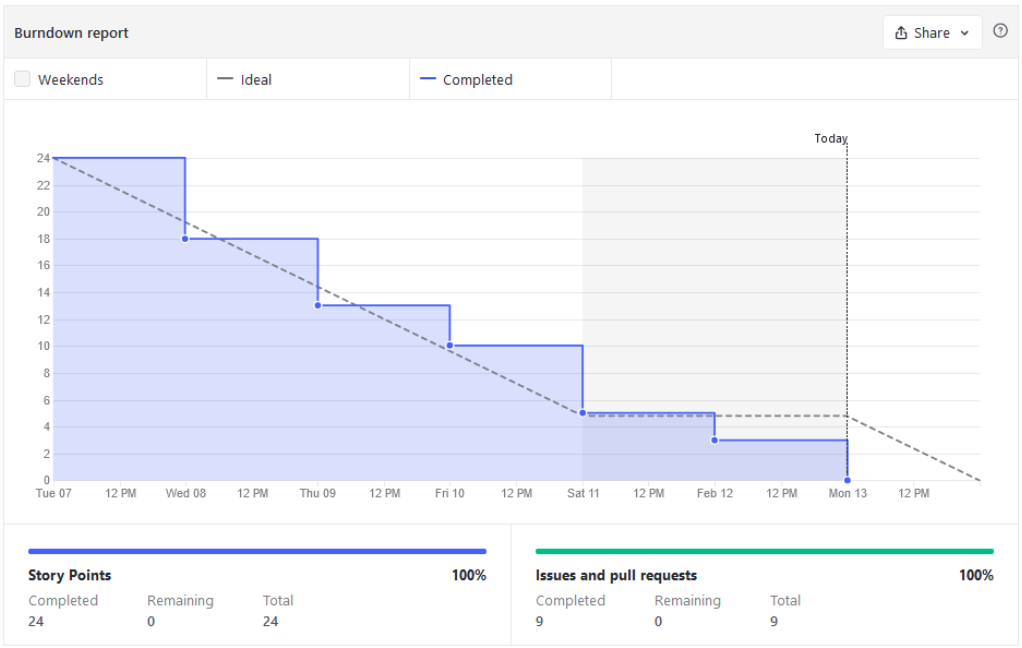
  1. Implementar el *democratic-co learning* y documentarlo en la sección teórica.
  2. Documentar la extracción de vectores de características de *phishing*, ataques *web* típicos, la sección teórica del *phishing*, los anexos e incluir las correcciones realizadas por el tutor.
  3. Renombrar el repositorio y adaptar la memoria a la nueva sección de *phishing*: modificar introducción, trabajos relacionados etc.
  4. Crear ilustraciones para mejorar la memoria, además de modificar las existentes con el fin de incluirlas en formato `pdf` para mejorar la calidad.
  5. Revisar la extracción de vectores de características para *phishing* (*features* 1-7).
  6. Vincular el repositorio con Sonarcloud e implementar control de calidad. Cambiar el archivo `README.md` e incluir nuevos *badges*.

- **Marcas temporales**

  El *sprint* fue desarrollado entre el 7 y el 13 de febrero de 2022.

- ***Burndown Report***

Figura A.14: *Burndown Report Sprint 13*



  En este *sprint* todos los *issues* fueron cerrados. Al igual que en ocasiones anteriores, aunque la estimación particular de cada tarea pudo no ser correcta, en general la de todo el *sprint* resultó equilibrada. Se calcula que el tiempo real dedicado al proyecto fue ligeramente superior, de unas 26 horas. Esto es debido principalmente a que se dedicó tiempo a la revisión «estilística» de la memoria (corrección de cursivas incorrectas, nombres propios, anglicismos, etc.).

- ***Sprint review meeting***

  Lo más destacable es la corrección de ciertos aspectos relacionados con el *democratic-co learning*. Por ello, se estimó que debían ser corregidos.

Sprint 14: Chimichurri

- ***Planning meeting***

  Se decidió trabajar en:

1. Corrección de los vectores de características de *phishing* y de la API de extracción de enlaces. Generación de *dataset* parcial y gráficas de pruebas para comprobar el nivel de detección de la propuesta.
2. Documentar algún aspecto pendiente. En especial herramientas nuevas, anexos y el proceso de minería de datos (realizar ilustraciones).
3. Introducir DeepSourceBot y mejorar la calidad del proyecto (corregir defectos de código). Proteger las ramas del repositorio para prevenir posibles conflictos con el *bot*.
4. Incluir Travis CI y generar una *build* de *tests*. Reorganizar el proyecto en estructura de paquetes.
5. Corregir el *democratic-co learning*, realizar la comparativa contra sslearn y documentar los resultados.
6. Lanzar los primeros experimentos de detección de *phishing* y corregir posibles *bugs*.

- **Marcas temporales**

  El *sprint* fue desarrollado entre el 14 y el 20 de febrero de 2022.

- ***Burndown Report***

  En este *sprint* se concluyó el trabajo propuesto. Debido a que hubo ciertas dificultades, se calcula que la estimación real del trabajo es de 33 horas.

- ***Sprint review meeting***

  En el *feedback* se estimó que los algoritmos de semisupervisado están correctamente implementados y se decidió avanzar otros aspectos del proyecto.

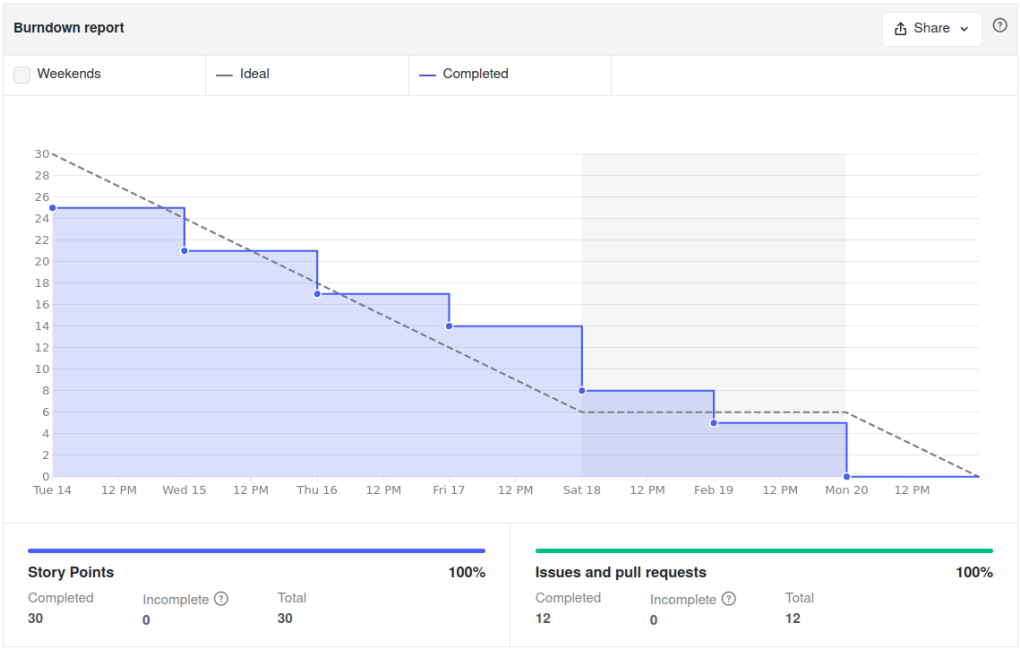
Sprint 15: Anise

- ***Planning meeting***

  Se decidió trabajar en:

  1. Comienzo de la *web*: aprender a utilizar Flask y desarrollar la aplicación. Manejar archivos `.css` básicos y aprender a usar (por encima) ciertas librerías como Bootstrap, SQL Alchemy, Flask WTF, etc. Búsqueda de una plantilla atractiva y desplegable.

Figura A.15: *Burndown Report Sprint 14*



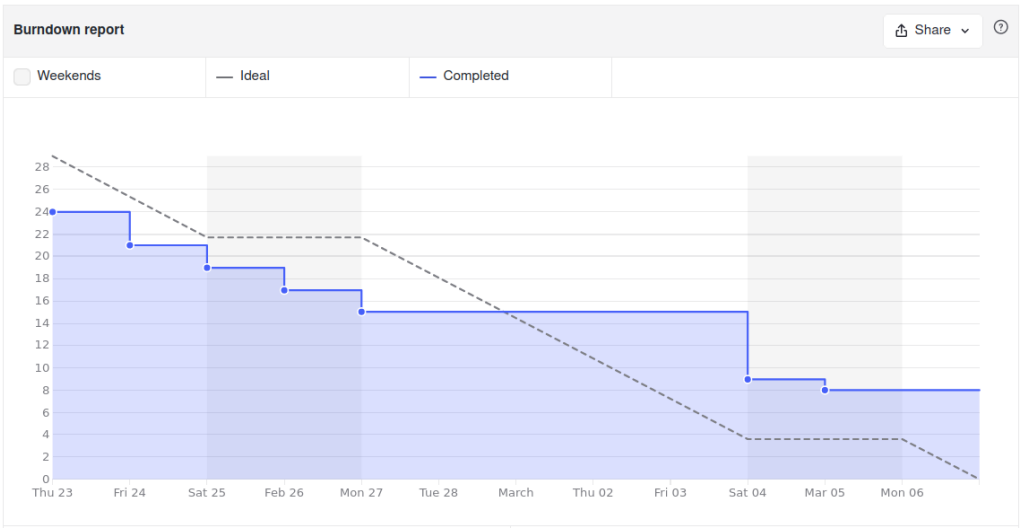
2. Sistema de bases de datos: buscar servidor (comparativa entre MariaDB y PostgreSQL) y desplegar en local.
3. Despliegue: despliegue en Heroku y gestión de errores. Levantar tanto la *web* como la base de datos.
4. Documentación: corrección de aspectos indicados por el tutor, redacción de anexos, inclusión del apartado «técnicas» en la memoria principal.
5. Experimentación: generación del *dataset* real de *phishing* (2 000 instancias) y realización de las gráficas con datos reales. Posible búsqueda de alternativas a la generación de vectores.
6. Calidad: corrección de *issues* utilizando Deep Source Bot.
7. Diagramas: comenzar a realizar diagramas de casos de uso, clases, etc.

- **Marcas temporales**

  El *sprint* fue desarrollado entre el 23 de febrero y el 6 de marzo de 2022.

- ***Burndown Report***

Figura A.16: *Burndown Report Sprint 15*



El tiempo estimado de trabajo fue de 29 horas. Sin embargo, el tiempo real ha sido de 37.5 horas, y no todos los *issues* pudieron ser cerrados. En concreto, se decidió posponer la búsqueda de nuevas formas de generar vectores de características por los buenos resultados reportados y no se dedicó tanto tiempo como el esperado a la generación de diagramas (únicamente se realizó el de casos de uso).

Claramente se subestimó el tiempo requerido en realizar pantallas *web*, desplegar en Heroku y gestionar servidores de bases de datos. Hubo complicaciones debido a que se ha escogido una plantilla profesional, y el uso del lenguaje es complejo. Todavía hay ciertos aspectos que arreglar respecto al despliegue en Heroku (no renderiza correctamente los menús desplegables a diferencia de «local»).

- ***Sprint review meeting***

  En esta reunión se concluyó que el incremento había sido desarrollado correctamente con la excepción de los diagramas (debido a que se dedicó poco tiempo). Por lo tanto, se ofreció retroalimentación con respecto a los mismos con el fin de volverlos a generar. También se recibió *feedback* relacionado con la documentación.

Sprint 16: Vanilla

- ***Planning meeting***

  En esta reunión se plantearon los siguientes objetivos:

1. Diagramas: repetir el diagrama de casos de uso partiendo de la retroalimentación proporcionada por el *product owner*. Realizar el diagrama de clases de la base de datos relacional y generar *mockups* de las interfaces para la página *web*.
2. *Web*: implementación de funcionalidad. Serialización y deserialización de clasificadores, generación de funciones de utilidades para modularizar el ML y diseño de la interacción entre elementos (comunicación en variables de sesión, formularios nuevos, etc.). Nuevas pantallas (carga con animaciones para tareas que requieran más tiempo), adaptación del modelo de datos de la *web* al nuevo diagrama de clases e introducción de roles y permisos. Pantallas asociadas al administrador (esqueleto realizado y adaptación del menú). Mejoras estilísticas (segmentos utilizados, usuarios activos, *badges* y más) y corrección de errores (encabezados y sesiones activas). Implementación definitiva de los modelos de la base de datos (sustituidos los anteriores por no ser suficiente para cubrir los nuevos requerimientos).
3. Documentación: corregir aspectos previos y realizar anexos. Introducir las secciones de CD/CI, Gitflow, Scrum y complementar las herramientas. Planificación temporal del *sprint*.
4. Herramientas: realizar una primera «toma de contacto» con `D3.js` y decidir si merece la pena introducir esta tecnología en la página *web*.

- **Marcas temporales**

  El *sprint* fue desarrollado entre el 7 de marzo y el 13 de marzo del 2023.

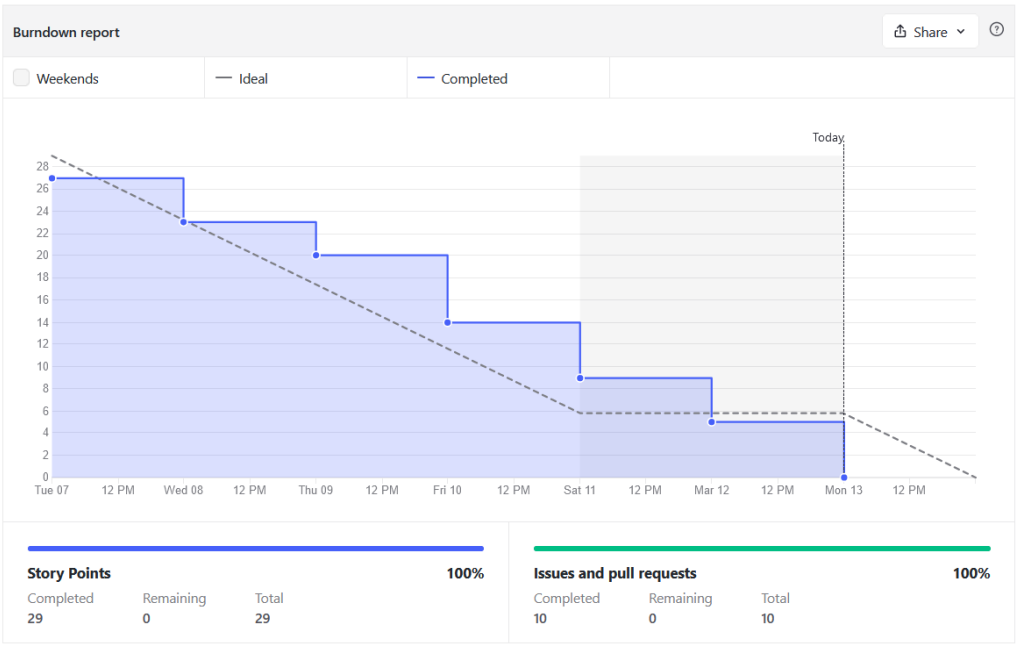
- ***Burndown Report***

  En este *sprint* todos los *issues* fueron cerrados. Se estimó un tiempo total de trabajo de 29 horas y el tiempo real dedicado ha sido de 30 horas. Cabe destacar que los *issues* de realización de *mockups* requirieron menos tiempo del estimado y que el tiempo sobrante se empleó en seguir programando la página *web*.

  Aunque no estaba planificado, también se incluyó la tecnología GitGuardian en el flujo de Git con el fin de proteger la aplicación ante posibles filtraciones de contraseñas (en concreto, de la base de datos).

- ***Sprint review meeting***

Figura A.17: *Burndown Report Sprint 16*



En esta reunión se concluyó que todos los *issues* fueron desarrollados correctamente a excepción de los diagramas (puesto que se realizó directamente un diagrama de «clases» para una base de datos en lugar de realizar la progresión correcta, es decir, entidad-relación, relacional, etc.). Por ello, se estimó conveniente repetir los diagramas.

Sprint 17: Sriracha

- ***Planning meeting***

  En esta reunión se decidió realizar un *sprint* más largo para avanzar en la funcionalidad general de la *web*. Para ello, se plantearon los siguientes objetivos:

  1. Diagramas: corregir pequeños aspectos del diagrama de casos de uso, realizar el diagrama entidad-relación y el relacional de la base de datos. Redactar el diccionario de datos.
  2. *Web*: implementación de funcionalidad vinculada con el nuevo modelo de datos (cambiar la base de datos anterior). Completar un *dashboard* funcional (con gráficas y que realice la extracción del vector de características, además de mostrar cómo se ha logrado y estadísticas de las URLs analizadas). Iniciar la administración

de modelos e implementar la interfaz principal y el formulario de creación de nuevos modelos desde la web (con funcionalidad, serializando los modelos generados en cada petición). Programar la interfaz principal de administración de instancias (consultas paginadas con formularios de *checkboxes*) y generar `csv` compatibles (y descargables) con el entrenamiento de modelos.

3. Herramientas: generar archivos de utilidades (refactorizar) y permitir la subida de archivos al servidor y la descarga desde el mismo. Corregir el *select all* del menú principal y hacerlo estético. Hacer comprobación de *emails* propia y eliminar la dependencia, además de asegurar que los formularios no dejen avanzar si no están completados los datos requeridos.
4. Documentación: corregir aspectos previos y realizar anexos. Documentar el despliegue en Heroku, revisar la memoria (en busca de posibles pérdidas de información), corregir el *mockup* del *dashboard* y repasar los trabajos relacionados.
5. Semisupervisado: revisar el *feedback* proporcionado por Álvar Arnaiz y César Ignacio García acerca de la inicialización del parámetro w en el *co-forest*. Corregir el código, repetir experimentos y documentar los resultados.
6. Calidad: corregir defectos de código.
7. Despliegue: desplegar el repositorio desde raíz (en lugar de sólo la *web*) y cambiar las dependencias (minimizando al máximo y solucionando incompatibilidades) del `requirements.txt`.

- **Marcas temporales**

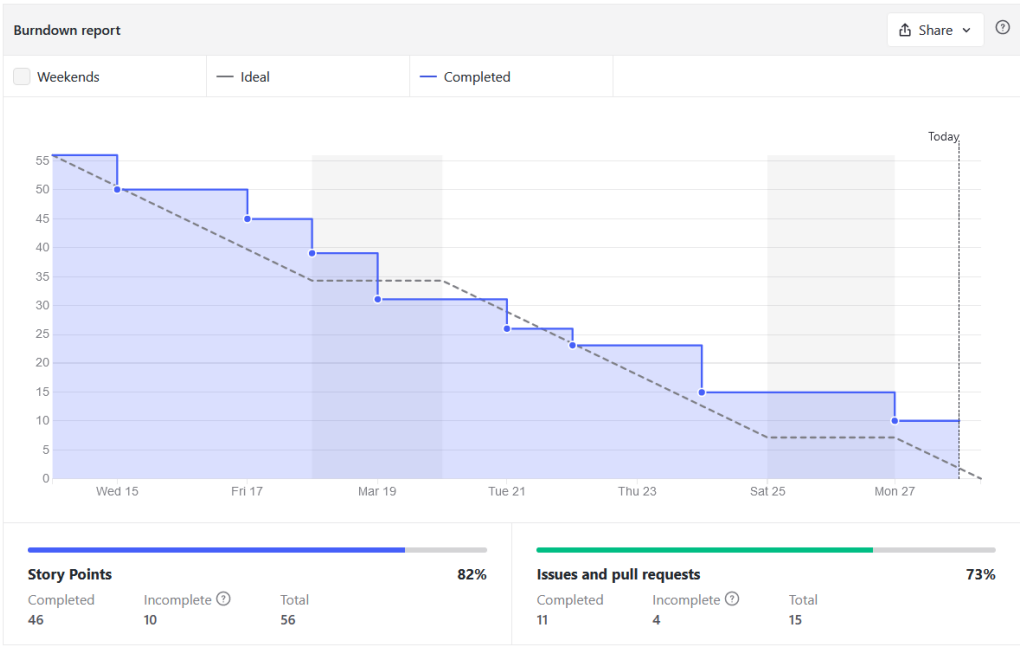
  El *sprint* fue desarrollado entre el 14 de marzo y el 28 de marzo del 2023.

- ***Burndown Report***

  En este *sprint* no se consiguió cerrar todos los *issues* planificados.

  Los diagramas y la redacción del diccionario de datos requirieron el doble de tiempo del estimado debido a las múltiples revisiones (con sus respectivas correcciones) por parte del tutor y otros profesores (Jesús Maudes). Además, se subestimó completamente el tiempo requerido en la realización de la *web* y, aunque se cerraron los *issues*, algunos de ellos se implementaron de forma «optimista» (falta «hacer segura» la implementación mediante comprobaciones de tipos, tratamiento de excepciones, etc.).

Figura A.18: *Burndown Report Sprint 17*



Por este motivo, no se cerraron los ítems 5, 6 y 7 y únicamente se documentaron los anexos (dentro del ítem 4). El tiempo real dedicado a la realización del *sprint* ha sido de 52.5 horas.

- ***Sprint review meeting***

  En esta reunión se revisó la nueva funcionalidad implementada. El *product owner* sugirió algunas mejoras dentro de cada pantalla mostrada. Las más destacables: internacionalizar la *web*, añadir nuevas funcionalidades (gráficos en el *dashboard*, pseudocódigos en la creación de modelos, etc). Además, se revisaron los diagramas realizados hasta el momento dándolos por válidos y se propuso empezar a documentar los nuevos.

Sprint N:

- ***Planning meeting***

  1.
  2.
  3.

4.

- **Marcas temporales**
- ***Burndown Report***
- ***Sprint review meeting***

A.4. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

La fase de análisis es una etapa fundamental en el ciclo de vida del desarrollo ya que permite entender los requerimientos del cliente e identificar los componentes necesarios para entregar un producto adecuado.

Durante esta fase, el equipo de desarrollo se reune con el cliente (o con su representante el *product owner*) y, tras realizar diversas entrevistas, se recoge lo que este espera de la aplicación. Además, el equipo de desarrollo analizará todos los requisitos no relacionados con la funcionalidad (como puede ser seguridad, escalabilidad, rendimiento, etc.) que se puedan deducir de dichas conversaciones.

Debido a la importancia que tiene esta fase, se ha experimentado una evolución con los años y se han propuesto distintas aproximaciones en función de las metodologías utilizadas. Entre ellas:

- **Metodologías tradicionales**: estas metodologías realizan la captura de los requerimientos en fases tempranas del desarrollo, realizando una «fotografía» exacta (e, idealmente, inmutable) de lo que el usuario necesita. Para ello, se hace uso de requisitos funcionales y no funcionales que se documentan en un formato estructurado y definido en estándares. Un ejemplo de especificación es el estándar IEEE 830 [1], donde se implora que los requisitos han de ser claros, precisos, medibles, coherentes, completos, factibles, específicos y verificables.
- **Metodologías ágiles**: en contraposición, las metodologías ágiles tratan de ser flexibles y adaptarse al usuario (ya que, muchas veces,

es común que no tenga claro en etapas tempranas de desarrollo lo que realmente necesita). Para ello utilizan las denominadas historias de usuario, que son breves descripciones de las funcionalidades que el usuario necesita para realizar una tarea específica. Estos documentos se recogen en estructuras tabulares y se almacenan en el *product backlog*, que lejos de ser una «fotografía», es una lista «viva» que se actualiza y prioriza en función de las necesidades del cliente.

Aunque en un proyecto real puede no resultar óptimo realizar una «mezcla» de metodologías, debido a las peculiares características de este (equipo de un desarrollador) y a que se solicita la inclusión de requisitos funcionales y no funcionales de una manera más tradicional en la memoria, se ha hecho uso de ambos métodos de documentación (complementados, además, mediante prototipos expuestos en la sección B.6). Por lo tanto, se realizará una especificación de requisitos clásica (expuesta en la sección B.5 y desarrollada en la sección B.7) complementada con alguna historia de usuario extraída de las entrevistas con el *product owner* (sección B.4).

Cabe destacar que durante el desarrollo real se ha utilizado una metodología ágil mediante el uso de *sprints* como se ha expuesto en la sección A.3.

B.2. Objetivos generales

En este proyecto se han definido distintos objetivos que pueden ser resumidos en los siguientes puntos:

1. Implementación y validación de algoritmos de aprendizaje semisupervisado: en concreto el *co-forest*, el *democratic-co learning* y el *tri-training*.

2. Aplicación del *machine learning* a la solución de un problema real relacionado con la ciberseguridad: se ha escogido la detección de *phishing* y detección de ataques en sistemas de recomendación, aunque esta última ha sido descartada por su bajo desempeño.

3. Desarrollo de una herramienta que permita poner al servicio de la comunidad el conocimiento desarrollado: se ha decidido desarrollar un analizador de *phishing* en formato página *web*, permitiendo además administración avanzada de modelos de *machine learning* e instancias de aprendizaje (URLs legítimos y fraudulentos).

Como se puede intuir, los dos primeros están enfocados a la investigación mientras que el último es una tarea de desarrollo. Por este motivo, este anexo se centrará principalmente en el tercer punto.

B.3. Usuarios

En las metodologías ágiles no sólo es necesario definir las funcionalidades, sino también quién las realiza. Por ello, se introducen a continuación los distintos usuarios de la página *web* desarrollada, aunque sus funciones completas se pueden observar en el diagrama de casos de uso B.11.

- **Visitante**: se trata de un navegante que accede a la *web*. En este caso, las funcionalidades que tiene disponibles son más limitadas, ya que únicamente se le permite analizar URLs y visualizar los resultados.
- **Usuario registrado**: un usuario registrado (rol «estándar») tiene acceso a funcionalidades colaborativas. Es decir, además de poder escanear URLS, tiene permiso para reportar enlaces que sepa que son legítimos o *phishing*. Estos serán revisados por administradores y etiquetados (de forma que pertenezcan a una lista blanca o a una lista negra). Además, también podrán notificar análisis incorrectos. De esta forma, si un usuario piensa que una URL es, por ejemplo, *phishing*, y los modelos predicen que se trata de un enlace legítimo, puede notificarlo. Este aviso llegará a los administradores, quienes podrán examinar la instancia e incluirla en el *dataset* para ser estudiada por modelos futuros y mejorar el rendimiento de la aplicación.
- **Administrador**: los usuarios que tengan este rol tendrán acceso a la funcionalidad completa de la *web*. Podrán crear nuevos modelos de aprendizaje y entrenarlos con las instancias que consideren, además de editar los existentes. También podrán revisar todas las notificaciones y reportes de usuarios de la aplicación e inluir nuevas instancias en el *dataset*, además de modificar las existentes.

B.4. Catálogo de historias de usuario

A modo ilustrativo y con el fin de realizar una memoria lo más completa posible (coherente con las metodologías ágiles), se facilitan en la tabla B.1 algunas historias de usuario tomadas durante las entrevistas con el *product*

ID	Como	Quiero	Para	Aceptación
HU-1	Visitante	Analizar URLs	Saber si un enlace es fraudulento (*phishing*) o legítimo	La página debe mostrar el resultado tras el análisis. La *web* debe informar si no es posible mediante un mensaje ilustrativo.
HU-2	Visitante	Poder registrarme	Crear una cuenta en la aplicación	La cuenta es persistente y se puede acceder a ella.
HU-3	Visitante	Poder iniciar sesión	Acceder a más funcionalidades	Al introducir credenciales correctas se accede a una cuenta y se desbloquean opciones colaborativas.
HU-4	Usuario registrado	Reportar URLs	Que sean incluídas en una lista blanca o negra	Existe un lugar donde reportar URLs cuando se está seguro de la clase a la que pertenecen.
HU-5	Usuario registrado	Notificar análisis incorrectos	Que las instancias sean revisadas y poder mejorar los algoritmos de ML	Existe un botón con el que se reporta automáticamente que se considera que un análisis ha fallado.
HU-6	Administrador	Administrar instancias (URLs)	Poder mantener un dataset actualizado	Se pueden añadir instancias. Se pueden generar ficheros `csv` con las instancias disponibles. Se pueden modificar instancias. Se pueden consultar las instancias reportadas por los usuarios.
HU-7	Administrador	Administrar modelos de *machine learning*	Que estén disponibles en la página y puedan ser utilizados por los usuarios para realizar análisis	Se pueden crear nuevos modelos desde la web. Se pueden probar modelos existentes. Se pueden editar modelos existentes.

Tabla B.1: Historias de usuario recogidas durante las entrevistas con el *product owner*

owner. Sin embargo, el catálogo completo de requisitos se encuentra en el apartado B.5.

B.5. Catálogo de requisitos

B.6. Prototipado

Como es conocido, en el mundo del desarrollo *software* suele haber una diferencia en el entendimiento de una aplicación por parte del cliente y del equipo de desarrollo que puede desembocar en malentendidos que causen retrasos temporales y pérdidas económicas.

Con el fin de reducir dichas desventajas y realizar un diseño que se adapte a los requerimientos del usuario, se ha decidido realizar una fase de prototipado durante las entrevistas del producto. Esto ha permitido identificar posibles diferencias entre el equipo del desarrollo y el cliente (representado por el *product owner*), facilitando la comunicación y colaboración entre ambas partes.

Se adjuntan a continuación los diferentes *mockups* del analizador de *phishing* realizados durante esta fase.

Figura B.1: Prototipos: página principal

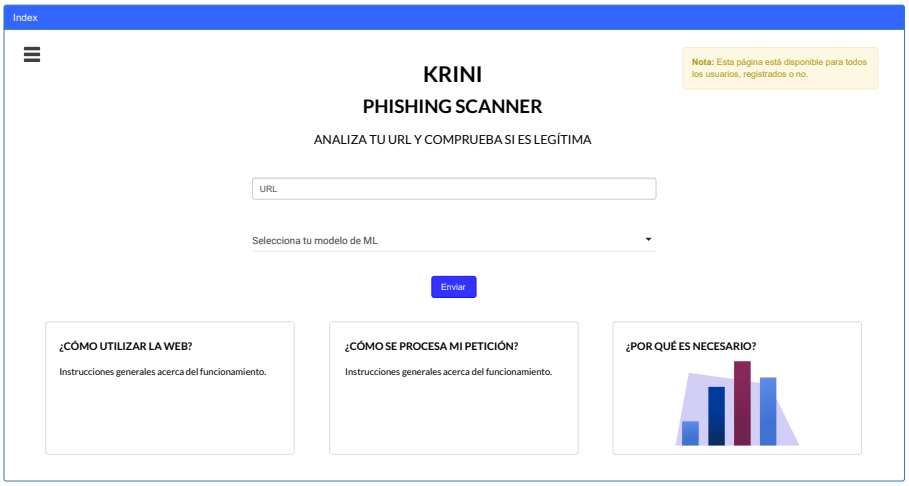
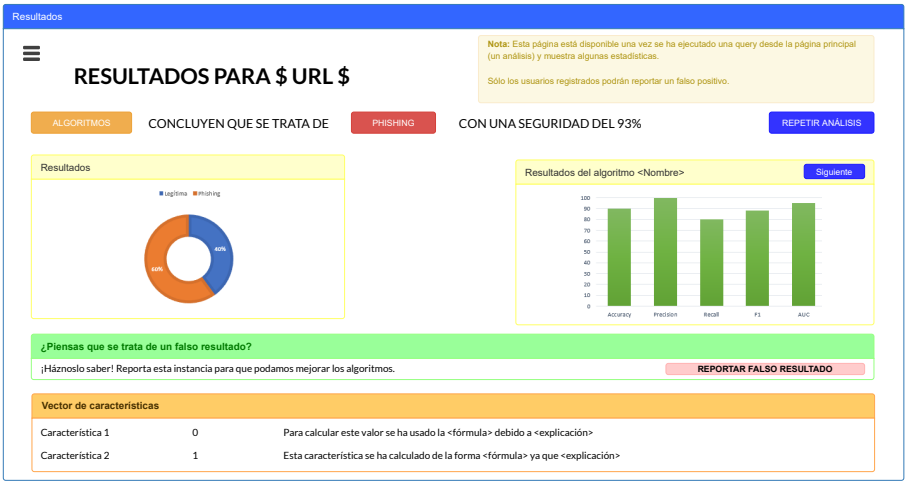


Figura B.2: Prototipos: *dashboard*



B.7. Especificación de requisitos

Se detalla a continuación la especificación de requisitos tradicional de la aplicación *web* propuesta basada en el catálogo de requisitos de la sección B.5.

Diagrama de casos de uso

Partiendo de las historias de usuario extraídas en las entrevistas realizadas con el *product owner* y de los *mockups* mostrados en la sección B.6, se ha extraído el diagrama de casos de uso disponible en la figura B.11.

A continuación se muestra la tabla correspondiente a cada caso de uso.

Figura B.3: Prototipos: reportar una URL

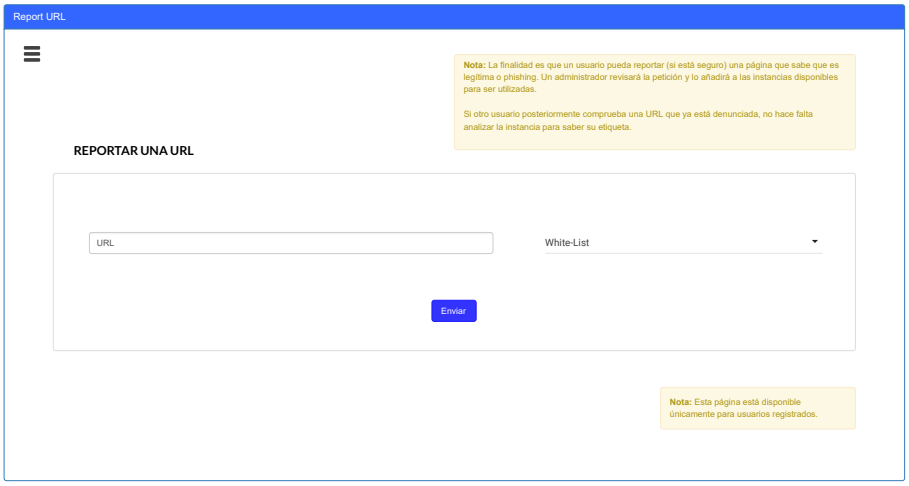


Figura B.4: Prototipos: perfil del usuario

Figura B.5: Prototipos: administración de modelos

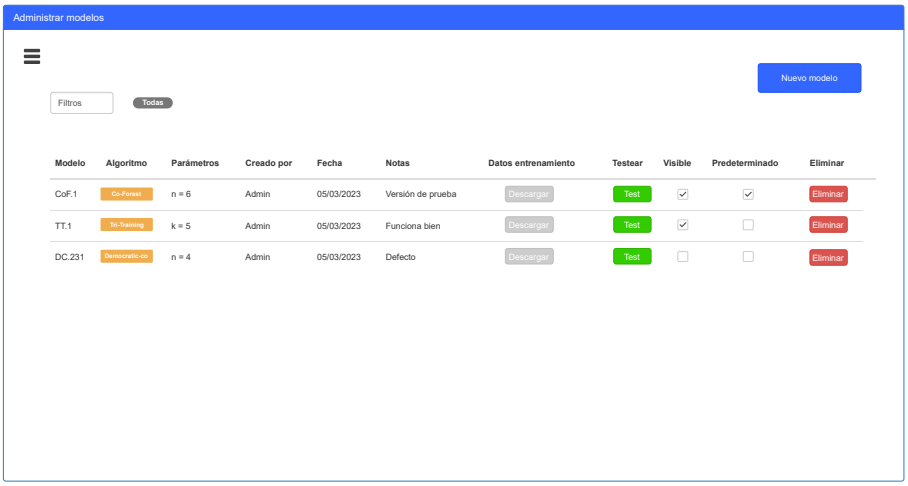


Figura B.6: Prototipos: creación de modelos

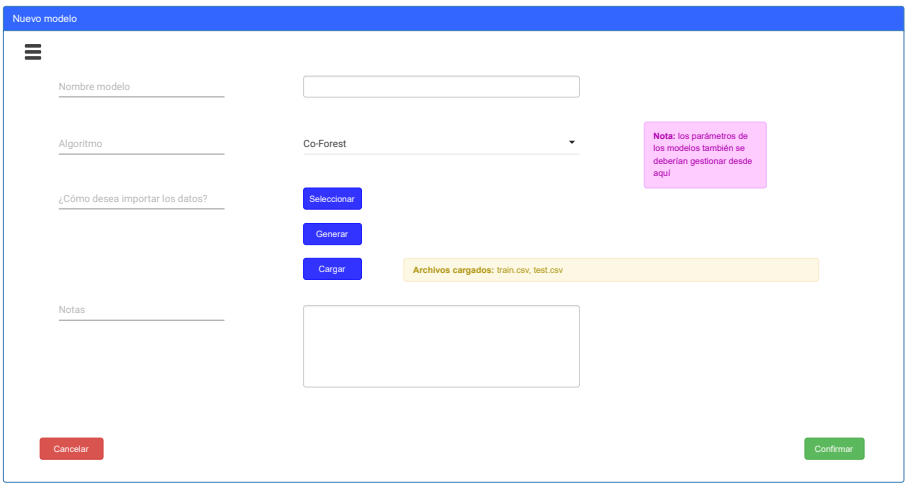


Figura B.7: Prototipos: evaluación de modelos

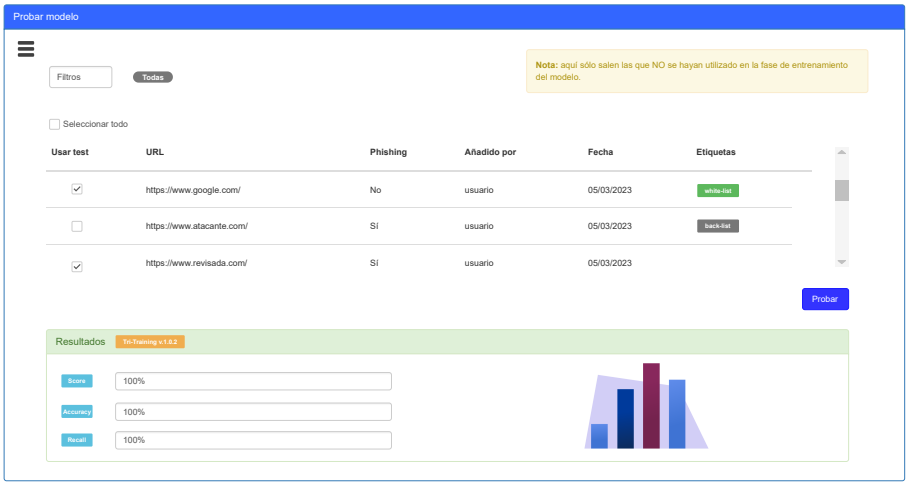


Figura B.8: Prototipos: selección de instancias

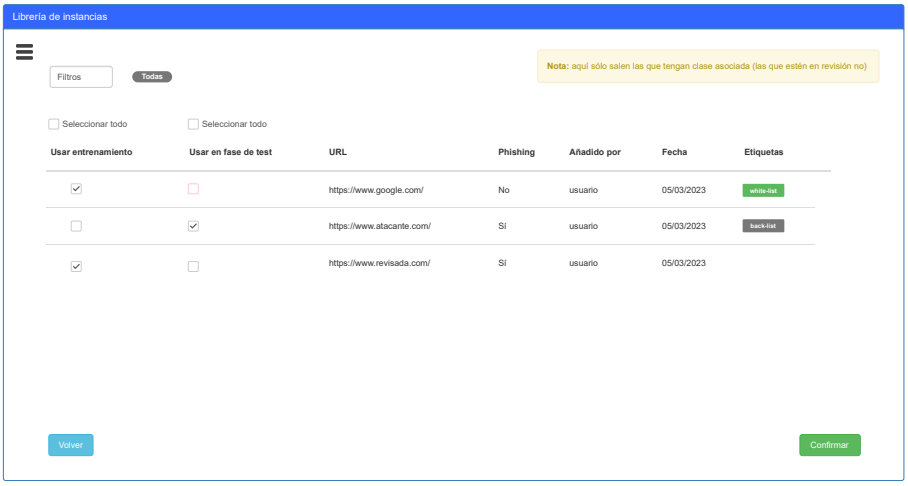


Figura B.9: Prototipos: administración de instancias

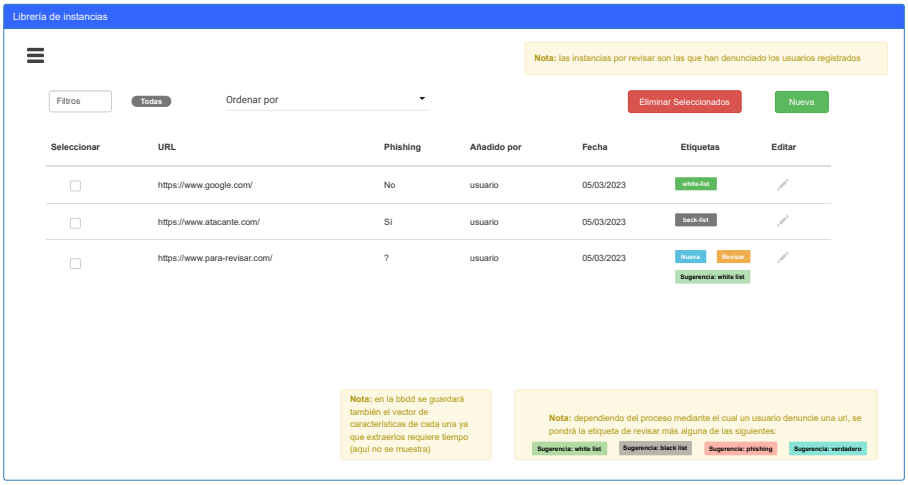


Figura B.10: Prototipos: edición de instancias

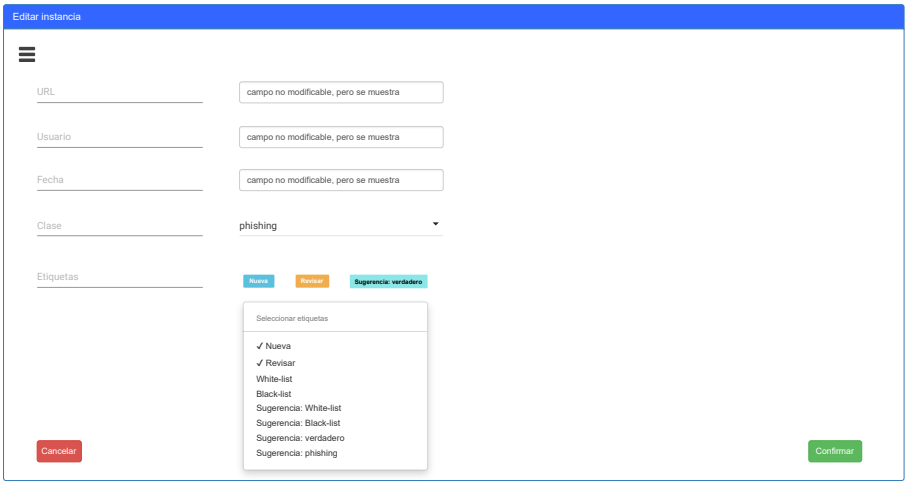
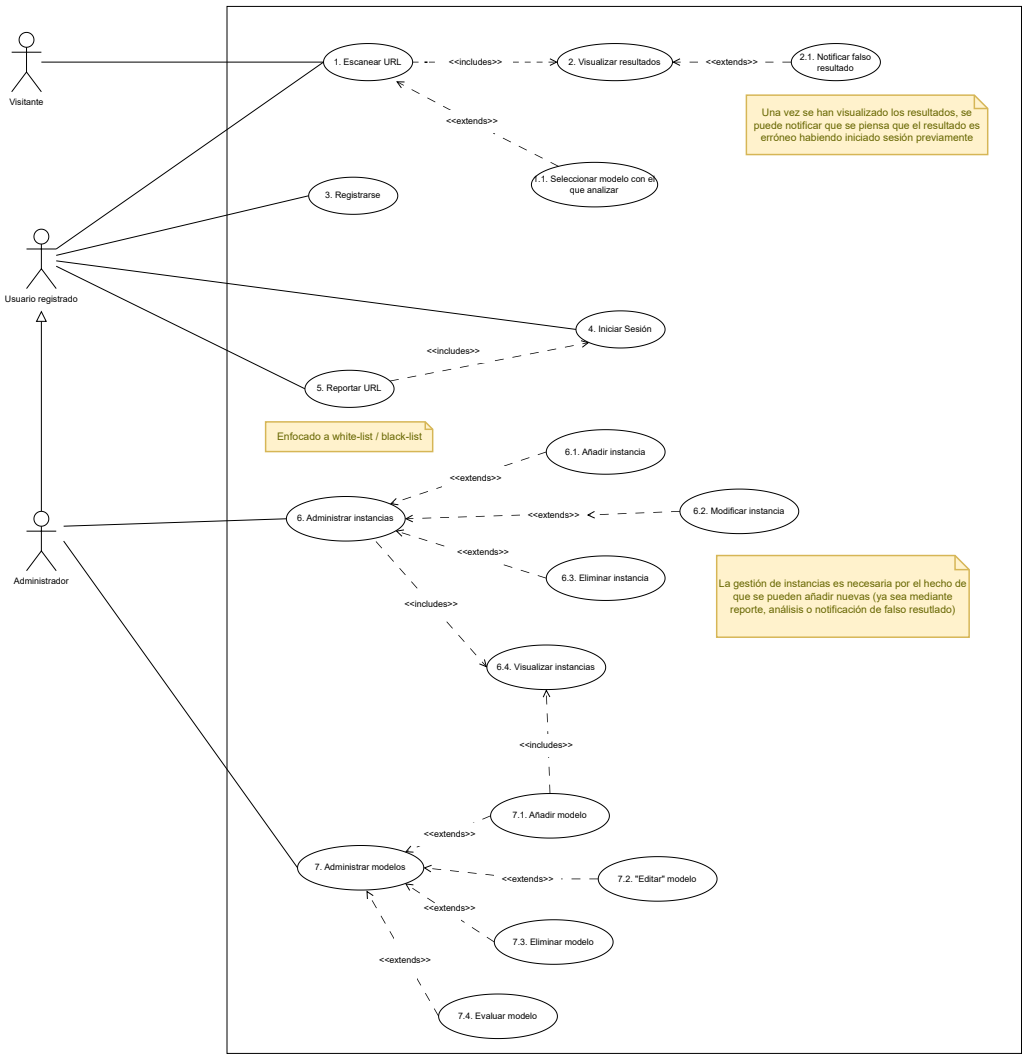


Figura B.11: Diagramas: casos de uso



CU-1	**Escanear URL**
Versión	1.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Permitir que un usuario realice un escaneo de la URL que desee. Para ello, seleccionará los modelos que considere adecuados y se mostrarán los resultados en un *dashboard* (CU-2 en la tabla B.4).
Precondición	No hay precondiciones.
Acciones	1. El usuario introduce la URL. 2. El usuario selecciona los modelos de ML que considere (consultar CU-1.1 en la tabla B.3). 3. El usuario decide si quiere realizar un análisis rápido. 4. El usuario pulsa el botón de «realizar análisis».
Postcondición	No hay postcondiciones.
Excepciones	En caso de que la URL introducida sea inalcanzable (tras aplicar reintentos) o que no haya ningún modelo disponible, el usuario será notificado y redirigido a la página principal.
Importancia	Alta

Tabla B.2: CU-1 Escanear URL.

CU-1.1	**Seleccionar modelos con los que analizar**
Versión	2.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Se mostrará al usuario los distintos modelos disponibles (y visibles) para que elija los que desee.
Precondición	Estar realizando el CU-1 (disponible en la tabla B.2).
Acciones	1. Se cargan los modelos disponibles en la base de datos. 2. El usuario selecciona los modelos que quiera utilizar.
Postcondición	Los modelos seleccionados quedan guardados.
Excepciones	En caso de no haber modelos disponibles la lista quedará vacía y se controlará la excepción en el CU-1 (tabla B.2).
Importancia	Baja

Tabla B.3: CU-1.1 Seleccionar modelos con los que analizar.

CU-2	**Visualizar resultados**
Versión	1.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Se muestra al usuario los resultados tras haber analizado la URL mediante los algoritmos de ML seleccionados. Para ello se hará uso de un *dashboard*, donde además se podrá notificar un falso resultado (CU-2.1 en la tabla B.5).
Precondición	Haber realizado un análisis previamente (CU-1 en la tabla B.2).
Acciones	1. El usuario es redirigido a un *dashboard*. 2. El usuario puede interactuar con los distintos gráficos y la información mostrada en la página. 3. El usuario puede notificar si considera que el análisis es erróneo (CU-2.1 en la tabla B.5).
Postcondición	No hay postcondiciones
Excepciones	En caso de intentar acceder al *dashboard* sin haber realizado un análisis previo, el usuario será redirigido a la página principal con un mensaje informativo.
Importancia	Alta

Tabla B.4: CU-2 Visualizar resultados.

CU-2.1	**Notificar falso resultado**
Versión	1.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Permite a un usuario notificar automáticamente a la aplicación si considera que el resultado de un análisis es erróneo.
Precondición	Encontrarse en el *dashboard* tras un análisis (CU-2 en la tabla B.4) y haber iniciado sesión (CU-4 en la tabla B.7).
Acciones	1. El usuario pulsa el botón «notificar falso resultado».
Postcondición	Se registra la URL notificada en la base de datos con el campo «sugerencia» establecido al valor opuesto del resultado mostrado.
Excepciones	Si el usuario no ha iniciado sesión, será informado de que la notificación no ha sido realizada por este motivo. En caso de errores con la base de datos se controlarán y se informará al usuario de la notificación no realizada.
Importancia	Baja

Tabla B.5: CU-2.1 Notificar falso resultado.

CU-3	**Registrarse**
Versión	1.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Permite a un usuario crear una cuenta en la aplicación.
Precondición	El usuario no debe estar registrado ni haber iniciado sesión (CU-4 en la tabla B.7).
Acciones	1. El usuario selecciona «Registrarse». 2. El usuario introduce el nombre de usuario que considere. 3. El usuario introduce un *email* asociado a la cuenta. 4. El usuario introduce la contraseña deseada. 5. El usuario pulsa el botón de «Registrar».
Postcondición	La información del nuevo usuario queda almacenada en la base de datos.
Excepciones	En caso de que los datos estén repetidos o el *email* sea incorrecto, se pedirá al usuario que corrija los errores y no se creará la nueva cuenta. Si hay un usuario *logueado* será redirigido a la página principal con un mensaje informativo.
Importancia	Media

Tabla B.6: CU-3 Registrarse.

CU-4	**Iniciar sesión**
Versión	1.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Permite a un usuario iniciar sesión en la aplicación.
Precondición	El usuario no debe haber iniciado sesión en el navegador (CU-4 en la tabla B.7).
Acciones	1. El usuario pulsa el botón de «Iniciar sesión». 2. El usuario introduce el nombre de usuario asociado a su cuenta. 3. El usuario introduce la contraseña asociada a la cuenta. 4. El usuario pulsa el botón de «Iniciar sesión».
Postcondición	La información del usuario queda cargada en las variables de sesión.
Excepciones	En caso de que los datos sean incorrectos el usuario será notificado. Si hay un usuario *logueado* será redirigido a la página principal con un mensaje informativo.
Importancia	Media

Tabla B.7: CU-3 Iniciar sesión.

CU-5	**Reportar URL**
Versión	1.0
Autor	Patricia Hernando Fernández
Requisitos asociados	RF-xx, RF-xx
Descripción	Permite a un usuario reportar una URL si considera que pertenece a una lista blanca o a una lista negra.
Precondición	El usuario debe de haber iniciado sesión (CU-4 en la tabla B.7).
Acciones	1. El usuario introduce la URL a reportar. 2. El usuario selecciona el tipo de lista a la que pertenece la URL. 3. El usuario pulsa el botón de «Enviar».
Postcondición	Se registra la URL reportada en la base de datos con el campo «sugerencia» establecido al valor seleccionado en el desplegable (ejemplo: *white-list*).
Excepciones	En caso de no haber iniciado sesión, el usuario será redirigido a una pantalla especial (error 403, acceso restringido) con un enlace para iniciar sesión. Se controla en aplicación que el usuario haya rellenado todos los campos necesarios.
Importancia	Baja

Tabla B.8: CU-5 Reportar URL.

Apéndice C

Especificación de diseño

C.1. Introducción

...

C.2. Diseño de datos

A continuación se expone el procedimiento seguido para desarrollar el diseño de datos de la aplicación.

Para implementar la persistencia, se ha decidido utilizar una base de datos PostgreSQL (debido a la compatibilidad con Heroku y a su caracter *open-source*). El diseño de la misma ha pasado por diversas fases: el diseño del modelo entidad-relación (disponible en la subsección C.2), el diseño del diagrama relacional (subsección C.2) y, por último, el diccionario de datos (subsección C.2).

Modelo entidad-relación

Para analizar las posibles relaciones entre entidades, se ha hecho uso del diagrama E-R disponible en la imagen C.1. Como se puede observar, ambas ISAs son exclusivas totales (es decir, todos los elementos pertenecen a una y solo una subclase).

Son destacables dos aspectos dentro del diagrama: la aparición de «bucles» y una relación ternaria. La relación ternaria se debe a la posibilidad de que un mismo usuario reporte más de una vez la misma URL en distintas fechas (se ha de mantener un historial). Por otro lado (y tras haberlo consultado

con el profesor Jesús Maudes), los «bucles» se han permitido debido a que, en este caso, los extremos aportan información y realizan roles muy distintos.

En el primero (relaciones «reporta» y «revisa»), un administrador debe revisar todas las instancias reportadas por cualquier usuario. En un lado, el rol desempeñado sería «reportar una URL», mientras que en el otro, sería «revisarla» y aceptarla (aporta nueva información, el usuario que reporta es distinto del que revisa). En el segundo (relaciones «administra», «revisa» y «se entrena») nuevamente hay una gran independencia en el rol desempeñado. Por ejemplo, para que un modelo se entrene con una instancia, debe de estar revisada por un administrador. Evidentemente este modelo va a ser gestionado por otro, pero es independiente de las acciones anteriores.

Diagrama relacional

Partiendo del modelo E-R disponible en la figura C.1, se ha realizado el diagrama relacional disponible en la imagen C.2.

A continuación se va a mencionar algunas de las decisiones de implementación tomadas. Sin embargo, el desarrollo completo de por qué se ha tomado cada una y todos los detalles se encuentran disponibles en el diccionario de datos de la sección C.2.

Respecto a la ISA existente en la entidad `Users`, se ha decidido implementar en una única tabla debido a que resulta favorecedor para los desarrolladores contar con el discriminante como campo de tabla y, además, únicamente hay un atributo extra para los usuarios «estándar» (que son la mayoría, por lo que la pérdida de espacio es completamente despreciable). En cuanto a la ISA en la entidad `Models`, se ha implementado considerando que es una interrelación 1:1 debido a que cada subclase tiene atributos muy distintos.

Por otro lado, se ha omitido la entidad `Date` ya que este es uno de los casos en los que no merece la pena que pase a ser tabla por no aportar información extra (resulta indiferente contar con una clase o con un *timestamp*). Aún así formará parte de la `PK` en la relación.

Para facilitar el desarrollo y reducir errores derivados de equivocaciones, se han decorado los atributos con un prefijo. De este modo, por ejemplo, el identificador de la tabla `Available_instances` pasará a llamarse `instance_id`. Además, también se ha añadido el prefijo *available* a ciertas clases para que sea más evidente la diferencia entre los propios objetos del sistema y la información almacenada en la base de datos.

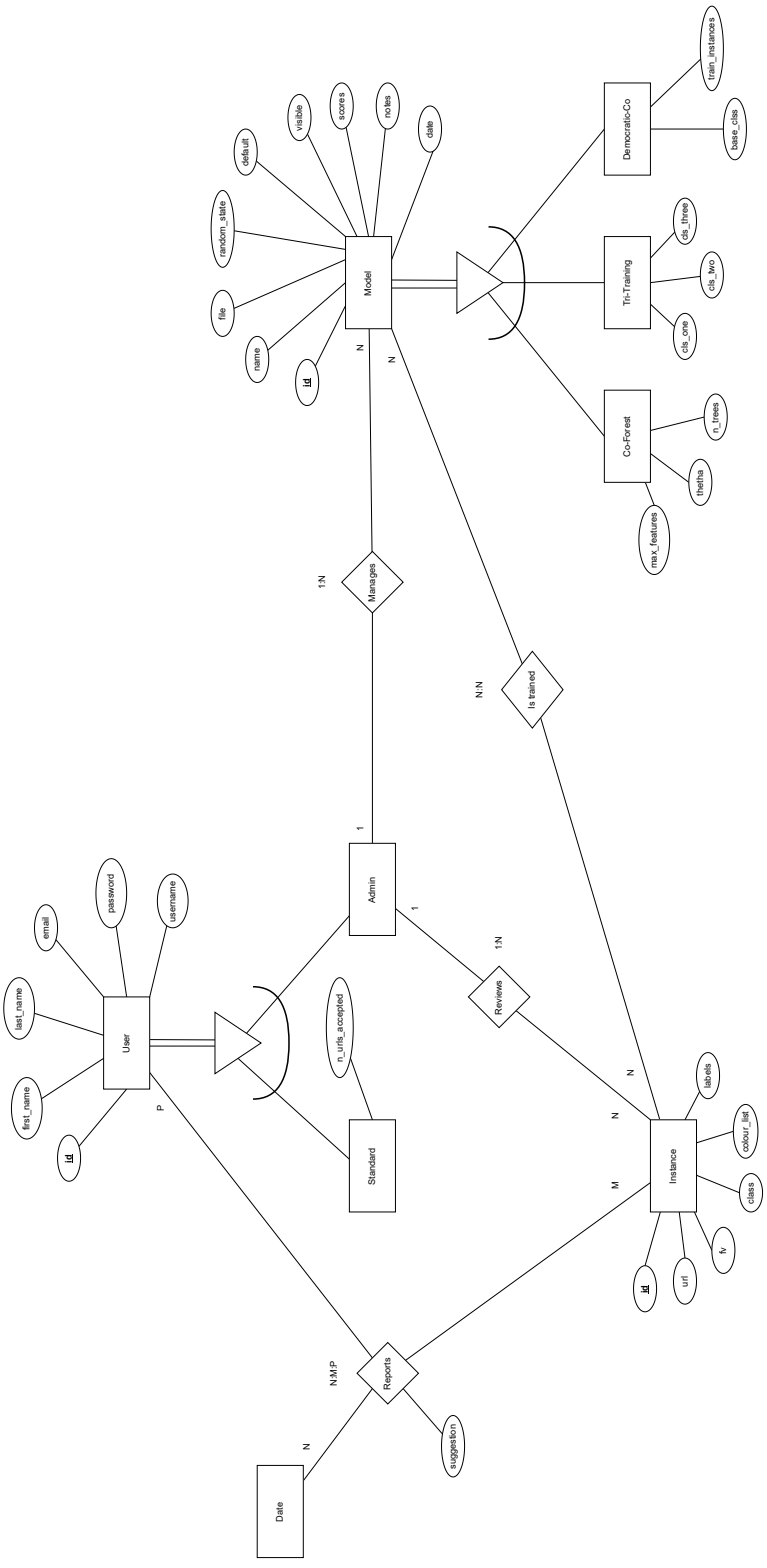


Figura C.1: Diagrama entidad relación de la aplicación

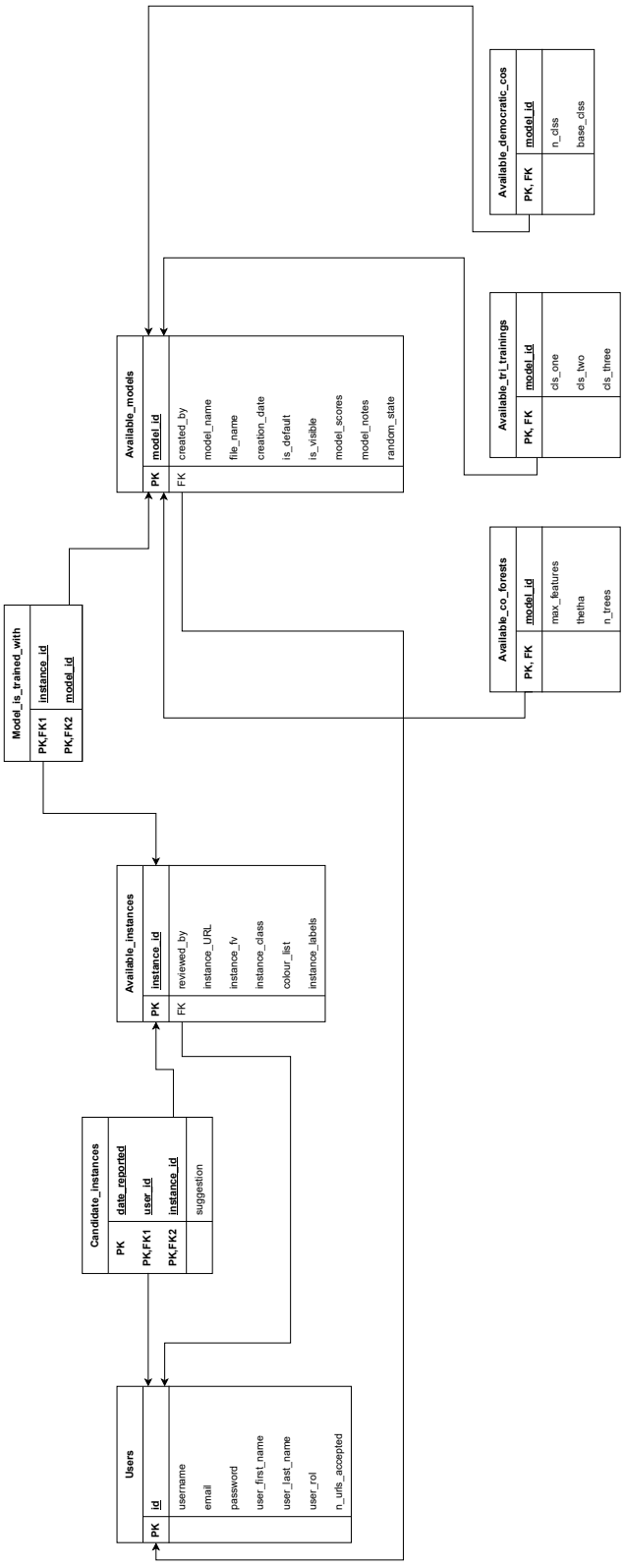


Figura C.2: Diagrama relacional de la base de datos

Nombre	Tipo	Columna	Descripción
id	INTEGER	PK	Identificador. Generado automáticamente mediante una secuencia.
username	VARCHAR(32)	UNIQUE NOT NULL	Inmutable. Nombre de usuario proporcionado en el registro.
email	VARCHAR(256)	UNIQUE NOT NULL	Inmutable. Email del usuario proporcionado en el registro.
password	BYTEA	NOT NULL	Contraseña cifrada mediante sha-512. La *salt* se obtiene aleatoriamente y está cifrada en sha-256.
user_first_name	VARCHAR(64)		Nombre del usuario.
user_last_name	VARCHAR(64)		Apellidos del usuario.
user_rol	VARCHAR(8)	NOT NULL	Rol del usuario. Puede ser «standard» o «admin». Por defecto: «standard».
n_urls_accepted	INTEGER		Atributo de usuarios «standard». Contador de aquellas instancias que han sido reportadas y aceptadas por un administrador.

Tabla C.1: Diccionario de datos. Tabla correspondiente a la clase Users.

Diccionario de datos

Se adjunta a continuación, para cada tabla, el diccionario de datos correspondiente.

Tabla de usuarios (Users)

La tabla C.1 almacena la información relevante de cada usuario registrado en la base de datos. Si se comprueba el diagrama E-R, se puede comprobar que una peculiaridad de esta tabla es que es una ISA. Debido a que, además del discriminante (usuario administrador o estándar), únicamente hay un atributo que diferencia ambas clases, se ha decidido implementar utilizando una sola tabla. Cabe destacar que la contraseña está correctamente cifrada mediante cifrada mediante sha-512 y la sal[1] se ha cifrado en sha-256.

Tabla de instancias (Available_instances)

La tabla C.2 almacena la información asociada a cada instancia disponible en el *dataset*. Las instancias son URLs que pueden ser utilizadas posteriormente para entrenar y probar modelos. Además, esta tabla también

[1] *Salt* es un término que se refiere a bits aleatorios usados como entrada (junto a la contraseña) en una función derivadora de claves (cuya salida es la contraseña cifrada).

Nombre	Tipo	Columna	Descripción
`instance_id`	`INTEGER`	`PK`	Identificador. Generado automáticamente mediante una secuencia.
`reviewed_by`	`INTEGER`	`FK(Users)`	En caso de estar vacío, significa que la URL todavía no ha sido aprobada por un administrador y está en proceso de revisión, por lo que no se puede utilizar para entrenar modelos.
`instance_url`	`VARCHAR(256)`	`UNIQUE NOT NULL`	URL de la instancia. Inmutable.
`instance_fv`	`INTEGER ARRAY[19]`		Vector de características de la URL.
`instance _class`	`INTEGER`		Etiqueta de la clase. Puede valer 0 si la URL es legítima o 1 en caso de que sea *phishing*.
`colour_list`	`VARCHAR(12)`		Puede ser «black-list», «white-list» o estar vacío. En caso de contener un valor ha de ser aprobada por un administrador.
`instance _labels`	`CHAR(128)`		Se preve que el campo varíe bastante, por lo que el tipo de dato no se ha definido como `VARCHAR`. Contiene etiquetas relacionadas con la instancia.

Tabla C.2: Diccionario de datos. Tabla correspondiente a la clase `Available_instances`.

almacena enlaces reportados por usuarios por pertenecer a una lista blanca o negra.

Tabla de modelos (`Available_models`)

La tabla C.3 almacena los modelos de aprendizaje que han sido creados por un administrador y están disponibles en la *web* para su uso.

Si se comprueba el diagrama E-R, se puede comprobar que esta tabla es una ISA. En este caso se ha decidido implementar considerando que es una interrelación 1:1. Por ello, se ha creado una tabla para la generalización y otra tabla para cada una de las especializaciones. Estas se enumeran a continuación:

- **`Available_co_forests`**: los modelos pertenecientes a este algoritmo poseen los atributos de la tabla C.4.
- **`Available_tri_trainings`**: los modelos pertenecientes a este algoritmo poseen los atributos de la tabla C.5.

Nombre	Tipo	Columna	Descripción
model_id	INTEGER	PK	Identificador. Generado automáticamente mediante una secuencia.
created_by	INTEGER	FK(Users) NOT NULL	Enlace al administrador que ha creado un determinado modelo. Dependencia por existencia.
model_name	VARCHAR(32)	UNIQUE NOT NULL	Inmutable. Nombre común y visible para los usuarios.
file_name	VARCHAR(32)	UNIQUE NOT NULL	Inmutable. Preferiblemente seguirá la sintáxis `nombre_v-a-b-c.pkl`, donde a, b, c son los números de versión (ejemplo: `cof_v-1-0-0.pkl`)
creation _date	TIMESTAMP		Fecha de creación.
is_default	BOOLEAN	NOT NULL	Por defecto «false». Determina si un clasificador es el utilizado por defecto (cuando un usuario realiza una consulta y no selecciona ningún modelo).
is_visible	BOOLEAN	NOT NULL	Por defecto «true». Determina si un clasificador está visible en la *web* y disponible para ser utilizado.
model_scores	FLOAT(3) ARRAY	NOT NULL	Por defecto `[0.0, 0.0, 0.0, 0.0, 0.0]`. Son las últimas estadísticas del último *test* realizado al clasificador. Las columnas (en este orden) corresponden a las métricas *accuracy*, *precision*, *recall*, F1 y ROC AUC.
model_notes	VARCHAR(128)		Notas acerca de los clasificadores base utilizados para crear el *ensemble*. No debería cambiar de valor.
random_state	INTEGER		Semilla aleatoria.

Tabla C.3: Diccionario de datos: tabla correspondiente a la clase `Available_models`.

Nombre	Tipo	Columna	Descripción
model_id	INTEGER	PK, FK(Available _models)	Identificador. Generado por la secuencia de la tabla padre e introducido mediante código (debe coincidir).
max_features	VARCHAR(4)	NOT NULL	Parámetro del árbol de decisión. Puede ser «sqrt» o «log2». Por defecto «log2».
thetha	FLOAT(3)	NOT NULL	Parámetro thetha del *ensemble* (umbral de confianza). Por defecto 0.75.
n_trees	INTEGER	NOT NULL	Número de árboles. Por defecto 3.

Tabla C.4: Diccionario de datos: tabla correspondiente a la clase `Available_co_forests`.

Nombre	Tipo	Columna	Descripción
model_id	INTEGER	PK, FK(Available_models)	Identificador. Generado por la secuencia de la tabla padre e introducido mediante código (debe coincidir).
cls_one	VARCHAR(4)	NOT NULL	Algoritmo del primer estimador base. Puede ser «kNN», «NB» o «tree».
cls_two	VARCHAR(4)	NOT NULL	Algoritmo del segundo estimador base. Puede ser «kNN», «NB» o «tree».
cls_three	VARCHAR(4)	NOT NULL	Algoritmo del tercer estimador base. Puede ser «kNN», «NB» o «tree».

Tabla C.5: Diccionario de datos: tabla correspondiente a la clase `Available_tri_trainings`.

Nombre	Tipo	Columna	Descripción
model_id	INTEGER	PK, FK(Available_models)	Identificador. Generado por la secuencia de la tabla padre e introducido mediante código (debe coincidir).
n_clss	INTEGER	NOT NULL	Número de clasificadores base.
base_clss	VARCHAR(4) ARRAY	NOT NULL	Array con los algoritmos pertenecientes a los estimadores base. Los valores que puede contener son «kNN», «NB» o «tree».

Tabla C.6: Diccionario de datos: tabla correspondiente a la clase `Available_democratic_cos`.

- **`Available_democratic_cos`**: los modelos pertenecientes a este algoritmo poseen los atributos de la tabla C.6.

Tablas de relación

Se enumeran a continuación las tablas que son resultado de una relación entre entidades.

- **`Candidate_instances`**: en la tabla C.7 se almacenan aquellas instancias que han sido reportadas por un usuario y necesitan ser revisadas por un administrador antes de ser incluidas en el *dataset* utilizable de la aplicación. Como se puede comprobar, se trata de una relación ternaria entre el usuario, la instancia reportada y la fecha (puede ser que un usuario reporte la misma URL más de una vez en dos fechas distintas). Es destacable que este es uno de los casos (excepción) en

Nombre	Tipo	Columna	Descripción
user_id	INTEGER	FK(Users)	Identificador del usuario que ha reportado la URL.
instance_id	INTEGER	FK(Available_instances)	Identificador de la URL reportada.
date_reported	TIMESTAMP		Fecha de envío del formulario.
suggestion	VARCHAR(16)	NOT NULL	Lo que el usuario denuncia que la URL es (ejemplo: «white-list»).
PK(user_id, instance_id, date_reported)			Clave primaria compuesta de la relación.

Tabla C.7: Diccionario de datos: tabla correspondiente a la clase Candidate_instances.

Nombre	Tipo	Columna	Descripción
model_id	INTEGER	FK(Available_models)	Identificador del modelo que ha sido entrenado.
instance_id	INTEGER	FK(Available_instances)	Identificador de la instancia utilizada en la fase de entrenamiento.
PK(model_id, instance_id)			Clave primaria compuesta de la relación.

Tabla C.8: Diccionario de datos: tabla correspondiente a la clase Model_is_trained_with.

los que no merece la pena que la entidad «Fecha» pase a ser tabla debido a que no posee ninguna característica relevante.

- **Model_is_trained_with**: en la tabla C.8 se almacena qué instancia ha sido utilizada para entrenar qué modelo en el sistema.

C.3. Diseño procedimental

En el desarrollo de *software*, el diseño procedimental se refiere a la definición de pasos ordenados para realizar una determinada funcionalidad. Se utiliza para crear métodos y se basa en la descomposición del problema en pequeñas tareas ejecutadas secuencialmente para alcanzar el resultado deseado. Para ello, se ha hecho uso de diagramas de secuencia. Estos muestran la interacción entre distintos componentes en un sistema.

Con el fin de facilitar la comprensión de ciertos métodos, se muestran a continuación los diagramas de las principales funcionalidades implementadas.

- **Página principal**: en el diagrama C.3 se representan los pasos seguidos en la página principal. Como se puede comprobar, el usuario realiza una petición y permanece esperando en la pantalla de carga correspondiente. Mientras tanto, el analizador procesa su petición.

  Para extraer el vector de características es necesario que la dirección introducida sea alcanzable. En caso de que no lo sea, se realizan correcciones (completar protocolos, corregir sintáxis, etc.). Posteriormente se intenta recuperar posible información existente en la base de datos acerca de dicha instancia (si pertenece a una lista blanca o negra, si ya se tiene almacenado su vector de características, etc.).

  Si la dirección no es alcanzable y no hay información previa, se devuelve al usuario un mensaje informativo (es una excepción, no se puede analizar ni mostrar nada).

  En el caso de que exista información previa acerca de dicha instancia (el vector de características está almacenado), se puede analizar (ya sea llamable o no). Este camino también es seguido en caso de que el usuario seleccione el «análisis rápido», ya que recuperar el vector de características de la base de datos no es temporalmente costoso.

  Por último, en caso de que la URL sea llamable y si no se ha recorrido ningún camino previo, se analiza utilizando el método de extracción de vector de características expuesto en la sección teórica de la memoria. Este proceso requiere unos 40 segundos.

  Si no ha ocurrido ninguna excepción, en cualquiera de las opciones se ha extraído el vecto de características. Por lo tanto, sólo queda analizar y mostrar los resultados haciendo uso de técnicas visuales (gráficos interactivos) en el *dashboard*, por lo que se redirige a esta pantalla. En caso de que ocurra alguna excepción, el usuario será devuelto a la página principal con un mensaje informativo.

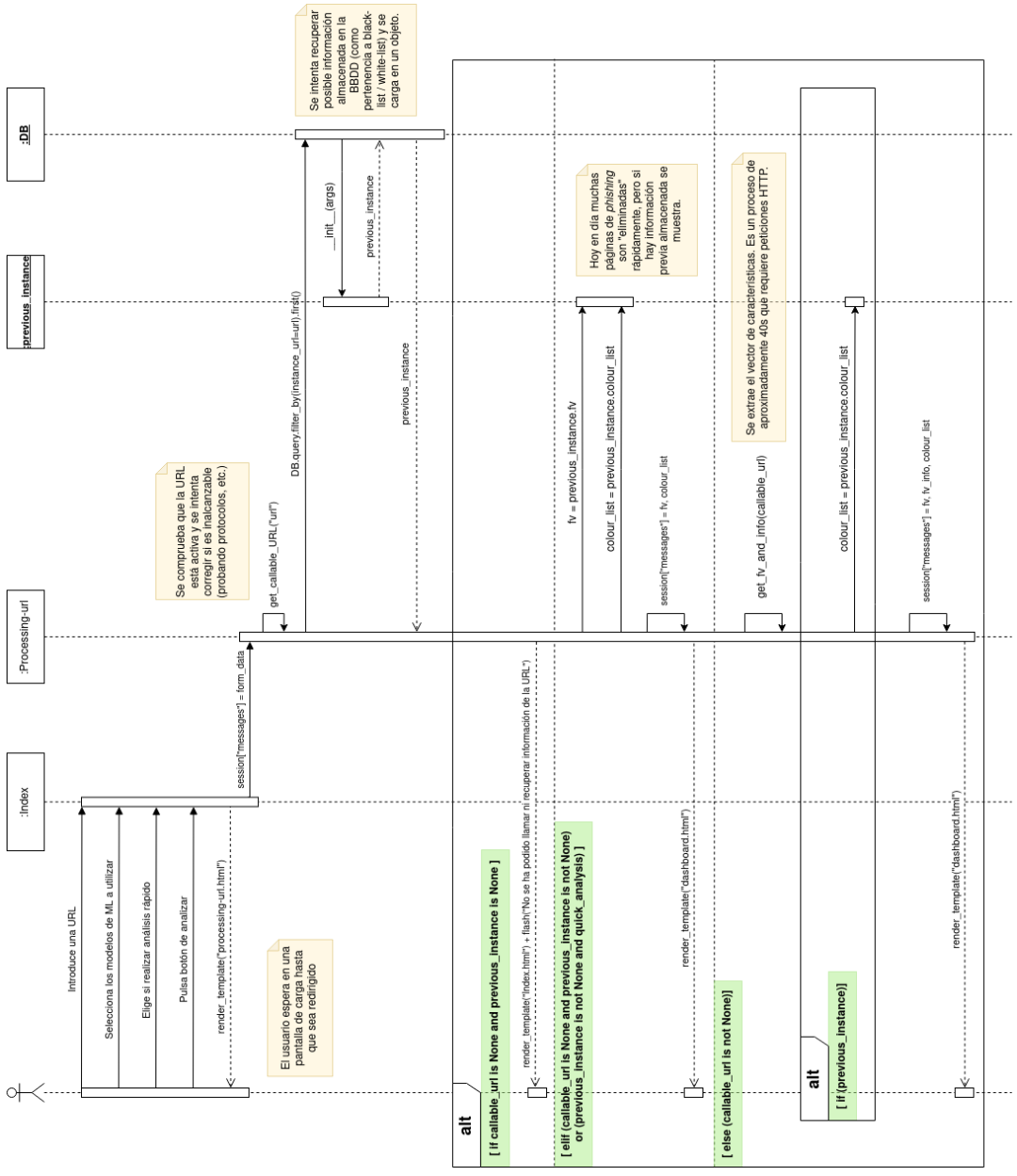


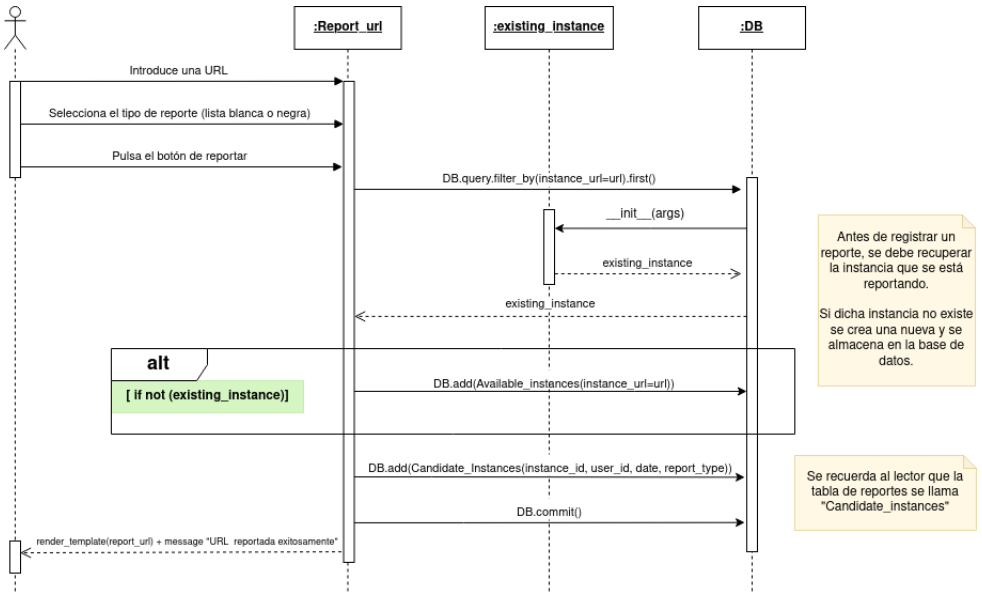
Figura C.3: Diagrama de secuencia de la página principal del analizador.

- **Página para reportar una URL**: la secuencia se ilustra en el diagrama C.4. Como se puede comprobar, el usuario tan solo ha de introducir su URL, seleccionar el tipo (lista blanca o lista negra) y aceptar.

  Como internamente la tabla donde se almacenan las URLs reportadas necesita información para incluir un nuevo registro (identificador de la instancia y del usuario), se intenta recuperar la instancia que se está reportando. En caso de no existir, se crea una nueva y se añade a la base de datos. Por último, se incluye el reporte con los datos y se redirige al usuario a la misma página con un mensaje informativo.

  Es destacable que un usuario registrado no va a poder acceder a esta pantalla (antes será redirigido por el sistema de control a la página de inicio de sesión).

Figura C.4: Diagrama de secuencia de la página para reportar una URL.



C.4. Diseño arquitectónico

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

D.4. Compilación, instalación y ejecución del proyecto

D.5. Compilación, instalación y ejecución de herramientas auxiliares

KEEL

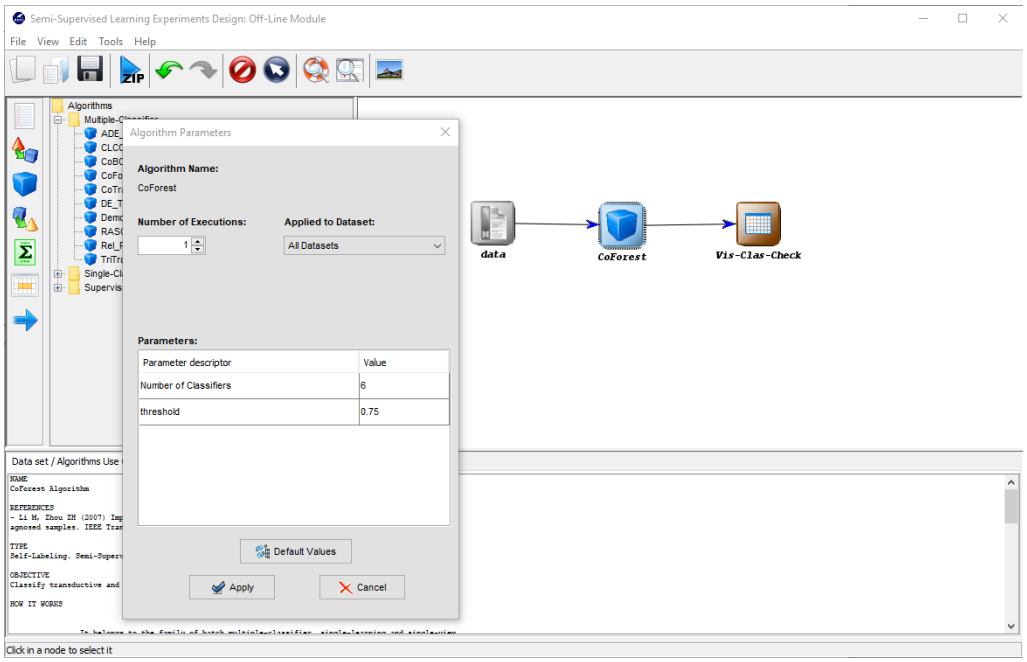
KEEL es una herramienta que permite experimentar con modelos de *machine learning*. Ha sido creada por distintas universidades españolas y financiada por el Ministerio de Educación y Ciencia [3].

Para poder ejecutarla, en primer lugar, se han de descargar los ficheros fuente del repositorio de GitHub [2]. Una vez se han descargado, se compilan aprovechando el fichero `build.xmlz` contenido y la herramienta `ant`.

Mediante el comando `ant cleanAll` se eliminan barios previos (para evitar conflictos), y mediante el comando `ant` se compila el código fuente.

Posteriormente se ejecuta la aplicación mediante el comando `java -jar ./dist/GraphInterKeel.jar` y se utiliza mediante su interfaz gráfica.

Figura D.1: Configuración de un experimento que utiliza el algoritmo *co-forest* mediante la GUI de KEEL.



D.6. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Bibliografía

[1] Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, 1998. https://ieeexplore.ieee.org/document/720574.

[2] Jesús Alcalá, Alberto Fernández, Julián Luengo, Francisco Herrera, Joaquín Derrac, Salvador García, and Luciano Sánchez. Github: Keel, 2018. https://github.com/SCI2SUGR/KEEL.

[3] Jesús Alcalá, Alberto Fernández, Julián Luengo, Francisco Herrera, Joaquín Derrac, Salvador García, and Luciano Sánchez. Keel: Knowledge extraction based on evolutionary learning, 2018. https://sci2s.ugr.es/keel/members.php.

[4] 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.

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

[6] Marta Palacio. *Scrum Master. Temario troncal 1.* 02 2022. https://www.scrummanager.com/files/scrum_master.pdf.

[7] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: Taxonomy, software and empirical study. *Knowledge and Information Systems*, 42, 02 2015.

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

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

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