UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Gºen Ingeniería en Informática

Trabajo final del GºIng.Informática:

Plataforma de experimentación de minería de texto sobre repositorios de código abierto GitLab

Presentado por Willow Maui García Moreno
en septiembre de 2020
Tutor Jesús M. Alonso Abad y Carlos López Nozal

Índice de contenido

Índice de ilustraciones

Índice de tablas

Apéndice A

I - Plan de Proyecto Software

. A.1. Introducción

En este apartado se expondrá la planificación temporal y el estudio de viabilidad, en el cual haremos un estudio de viabilidad económica y uno de viabilidad legal.

. A.2. Evolución temporal

La planificación temporal se ha llevado a cabo en GiHub través de sprints medidos con los hitos. El repositorio del proyecto, en el apartado en el que se ha llevado a cabo es: https://github.com/wgm1001/Trabajo-de-Fin-de-Grado-Plataforma-de-text-mining-sobre-repositorios/milestones?state=closed

. Sprint 1: Tareas previas al desarrollo (Inicio a 18 de junio)

Esta fue la primera fase del proyecto, en la cual fue utilizada para el aprendizaje de los conocimientos necesarios para realizar el proyecto. También se instaló la base de datos.

Además de hacer los primeros prototipos de extracción y almacenamiento.

Las tareas encuadradas en este sprint:

- Probar el almacenamiento de datos en la base de datos
- Probar la extracción de issues de un repositorio
- Instalar base de datos
- Lectura del artículo mentado en la descripción del TFG

. Sprint 2: Fase de extracción y almacenamiento (19 de junio a 26 junio)

Este Sprint se dedicó a, con los conocimentos adquiridos en la fase anterior gracias a las pruebas, a desarrollar las clases relativas a la extracción y almacenamiento de las tareas.

Las tareas de este sprint:

- Crear estructura inicial de las clases relacionadas con la extracción
- Crear el Esquema de la base de datos

. Sprint 3: Crear modelo de predicción a partir del texto de la issue (26 de junio a 2 julio)

En este milestone se probó la extracción , se crearon y modificaron las tablas de la base de datos, se crearon los objetos Repositorio, Issue y Label. También se realizaron pruebas sobre estas estructuras y se arreglaron errores.

Las tareas de este Sprint:

- Crear estructura de clases del modelo.
- Decidir si usar prepared statement o dictionary.
- Hacer una prueba del modelo con scikitLearn.
- Solucionar problema con listas al sacarlas a la base de datos.
- Crear clases Issue y Label.
- Probar estructura de clases de almacenamiento.
- Probar estructura de extracción de datos previa.
- Modificar la extracción de datos para recoger nuevas características
- Crear tablas en la base de datos
- Crear tabla en la base de datos (duplicada)

. *Sprint 4: Pruebas y modelo inicial (3 de julio a 14 de julio)*

En este sprint el grueso del esfuerzo se centró en realizar pruebas para comprender el funcionamiento de las clases relacionadas con la clasificación de etiquetas.

También se actualizó las pruebas para utilizar doctest y convertirlas en automáticas.

Además de crear un repositorio de ejemplo en Gitlab para realizar estas pruebas.

Las tareas de este milestone:

- Realizar prueba sobre las clases del modelo.
- Actualizar los test para el uso de Doctest.
- Crear repositorio de ejemplo en Gitlab.

. Sprint 5: Comunicación con el usuario (15 de julio a 17 de julio)

Este Sprint fue especialmente duró, ya que se trató de finalizar la primera versión del programa, por lo cual hubo un sobresaturación de trabajo. En esta frase se adquirieron los conocimientos relativos al desarrollo de la interfaz gráfica, de la arquitectura de cliente-servidor con el uso de Flask, los formularios y programación web con HTML y CSS.

Las tareas de este milestone son:

- Crear tabla en la base de datos para los modelos.
- Arreglar problemas de concurrencia entre sesiones.
- Crear CSS.
- Añadir etiqueta details con información del modelo.
- Modificar funcionamiento de los transcriptores.
- Crear páginas referentes a la predicción de datos
- Realizar pruebas con Flask
- Aplicar programación devensiva para comprobar determinados argumentos

. Sprint 6: Arreglar defectos y mejorar (18 julio a 23 de agosto)

En este Sprint se descansó, aún así se adecuó todo para la persistencia de los modelos y se trataron de solucionar problemas que empezó a dar la base de datos tras actualizar el conector.

Las tareas de este Sprint:

- Fueron Adecuar clases para guardar modelos entrenados
- Solucionar problema de tipos en la base de datos

. Sprint 7: Seguir mejorando (24 de Agosto a 6 de septiembre)

En este hito se arreglaron diversos problemas de la base de datos, se movió a Docker y se creó un script para la creación de las tablas.

Se mejoró la interfaz gráfica, se implementó un log de errores y se modificaron los formularios.

Las tareas de este milestone son:

- Cambiar CSS.
- Implementar log de errores.
- Crear instancia de Docker con base de datos.
- Solucionar problema al guardar modelos con más de un repositorios
- Solucionar problemas con la base de datos
- Mover parámetros de base de datos a archivo

. Sprint 8: final (7 de septiembre a 21 de septiembre)

En este sprint se modificó la predicción para integrar el entrenamiento de un modelo por etiqueta de manera manual, pero se descartó tras las pruebas de desempeño y se eliminó la funcionalidad. Se solucionaron todos los fallos que quedaban, se implementó la computación paralela, y se modificaron diversas clases. Además se acabaron los manuales de instalación y uso.

Las tareas de este sprint son:

- Eliminar modelo por etiqueta
- Realizar pruebas de eficiencia
- Hacer que se muestren todos los datos del modelo a entrenarlo
- Solucionar problemas con modelo OneVsTheRest
- Cambiar conector de la base de datos al oficial
- Implementar computación paralela para entrenarlo
- Modificar predicción
- Solucionar que se necesite un formularios
- Realizar manual de instalación
- Sacar parametros de la base de datos a archivo.

. *A.3. Estudio de viabilidad*

En este apartado se estudiará tanto la viabilidad económica, haciendo una estimación de los costes de producción y mantenimiento y la viabilidad legal mostrando las licencias de las dependencias.

. Viabilidad económica

Para este apartado se dividirán los costen en función de su naturaleza.

A. Costes humanos

Aquí estimaremos los costes que podría haber generado el tener contratada a la persona que lo ha desarrollado.

Suponiendo un salario Bruto de 1250 € mensuales con una duración de 3 meses con una jornada de 20 horas semanales.

Concepto	Coste
Salario mensual bruto	1.250,00 €
Retención del IRPF	207,75 €
Seguridad Social	952,50 €
Salario mensual neto	964,90 €
Coste total 3 meses	3.750,00 €

Tabla 1: Costes humanos

B. Costes del Producto

En este apartado se mostrarán los costes de mantener este programa en producción.

Teniendo en cuenta que se trataría de alojar de manera online se contrataría el plan de *production* en Heroku, lo cual tiene un coste 20 USD mensuales. Para el desarrollo ha sido necesario un ordenador, el cual tiene un sistema operativo de Windows. Si bien el equipo utilizado tiene otras características supongamos un equipo de 600 €, ya que no es necesario el equipo en concreto que se ha utilizado.

Concepto	Coste
Servidor online	21,49 €/mes
Licencia Windows 10 pro	137,77 €
Equipo informático	600,00 €

Tabla 2: Costes del producto

. Viabilidad legal

En este apartado se expondrán las licencias de las dependencias del programa. Como podemos ver todas las licencias son de código abierto para su uso.

Dependencias	Licencia
mysql-connector-python	GNU GPL o de Uso Comercial si se desea incorporar en el producto
scikit-learn	licencia BSD de 3 cláusulas
nltk	Apache 2.0
Flask	BSD
WTForms	Licencia de MIT
Flask-WTF	BSD
python-gitlab	LGPL v3.0

Apéndice B

II - Especificación de Requisitos

. B.1. Introducción

En este apéndice se expondrán los objetivos generales del proyecto , los requisitos funcionales y la especificación de los mismos.

. B.2. Objetivos generales

El objetivo de este proyecto es el desrrollo de una aplicación que extraiga las tareas de un repositorio de GitLab y con ellas entrene un modelo capaz de etiquetar nuevas tareas.

. B.3. Catalogo de requisitos

Expondremos los requisitos funcionales de la aplicación:

- RF-1 La aplicación ha de ser capaz de extraer las tareas de un repositorio y almacenarlas
  - RF-1.1 El usuario deberá ser capaz de aportar su propio Token de acceso a Gitlab
- RF-2 La aplicación deberá ser capaz de entrenar modelos de clasificación con los repositorios extraídos
  - RF-2.1 Deberán poder ser modificados los parámetros de entrenamiento de los modelos
  - RF-2.2 Deberán poder ser almacenados y recuperados los modelos entrenados.

. B.4. Especificación de requisitos

En este apartado detallaremos el diagrama de Casos de uso, Ilustración 1, para los anteriores requisitos.

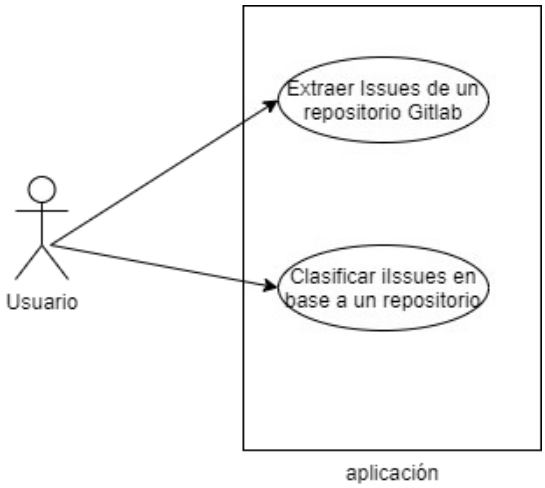

Ilustración 1: Diagrama de casos de uso

Caso de uso 1:	Extraer Issues de un repositorio	
Descripción	Se han de extraer las características necesarias para el posterior entrenamiento del modelo. Se debe poder propiciar un Token de acceso a GitLab propio.	
Requisitos	RF-1 RF-1.1	
Precondiciones	Ha de existir el proyecto a extraer Ha de estar correctamente configurado el programa y el Token ha de tener los permisos suficientes	
Secuencia normal	Paso	Acción
	1	Introducir link del repositorio del que se busquen extraer las Issues
	2	Dar al botón para extraerlas.
	3	Dar al boton de volver a Inicio una vez finalizada.
Excepciones	1	En el caso 1 se podrá adjuntar un Token de acceso personal
	2	En caso de no cumplir los prerequisitos saltará una pantalla de error especificando el error.
Postcondiciones	Las Issues estarán almacenadas en la base de datos.	
Importancia	Alta	
Urgencia	Alta	

Tabla 3: Casos de uso extracción

Caso de uso 2:	Entrenar Modelo	
Descripción	Se ha de entrenar un modelo acorde a unos parámetros personalizados, con al menos un repositorio.	
Requisitos	RF-2 RF-2.1 RF-2.2	
Precondiciones	Ha de haberse extraído algún repositorio Ha de seleccionarse al menos un repositorio	
Secuencia normal	Paso	Acción
	1	Introducir parámetros necesarios para entrenar
	2	Dar al botón para extraerlas.
	3	Dar al botón de volver a Inicio una vez finalizada.
Excepciones	1	En el caso 1 se cargar un módelo previamente entrenado.
	2	En caso de no cumplir los prerequisitos habrá que volver a la pestaña anterior.
	3	En caso de querer seguir prediciendo con el mismo modelo se puede repetir el paso 2.
Postcondiciones	La predicción aparecerá en la pantalla del paso 2 y el modelo habrá sido almacenado en la base de datos.	
Importancia	Alta	
Urgencia	Alta	

Tabla 4: Casos de uso de clasificación

Apéndice C

III - Especificación de diseño

. Introducción

En este anexo se presentan el diseño de los datos, el diseño procedimental y el diseño arquitectónico de la aplicación.

. Diseño de datos

Los Repositorios, las Tareas y las etiquetas están organizados en la base de datos como en la Ilustración 2 . Los modelos se guardan sin relaciones.

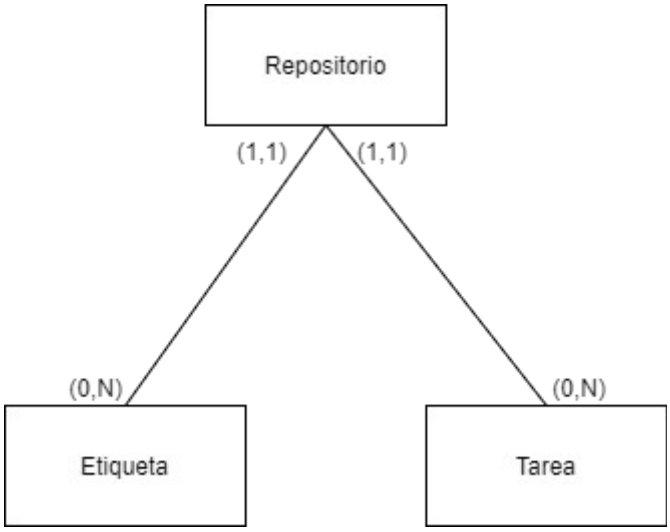

Ilustración 2: Diagrama de entidad relación

Ahora se expondrá el diagrama de clases de la aplicación, el cual está en la Ilustración 3.

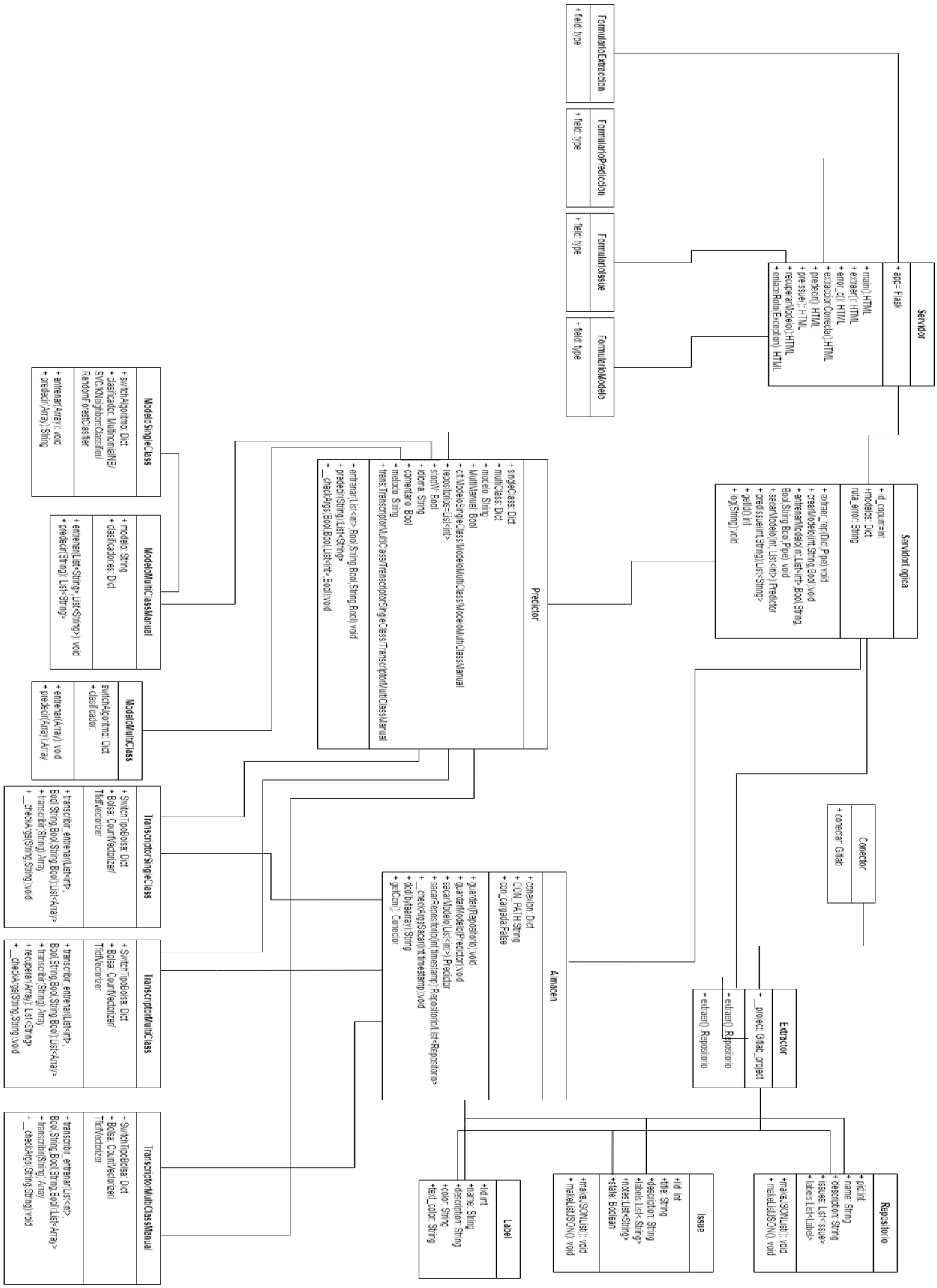

Ilustración 3: Diagrama de clases

. *Diseño procedimental*

En este apartado Mostraremos los diagramas de secuencia de la aplicación.

El diagrama de secuencia de la extracción de repostiorios es la Ilustración 4.

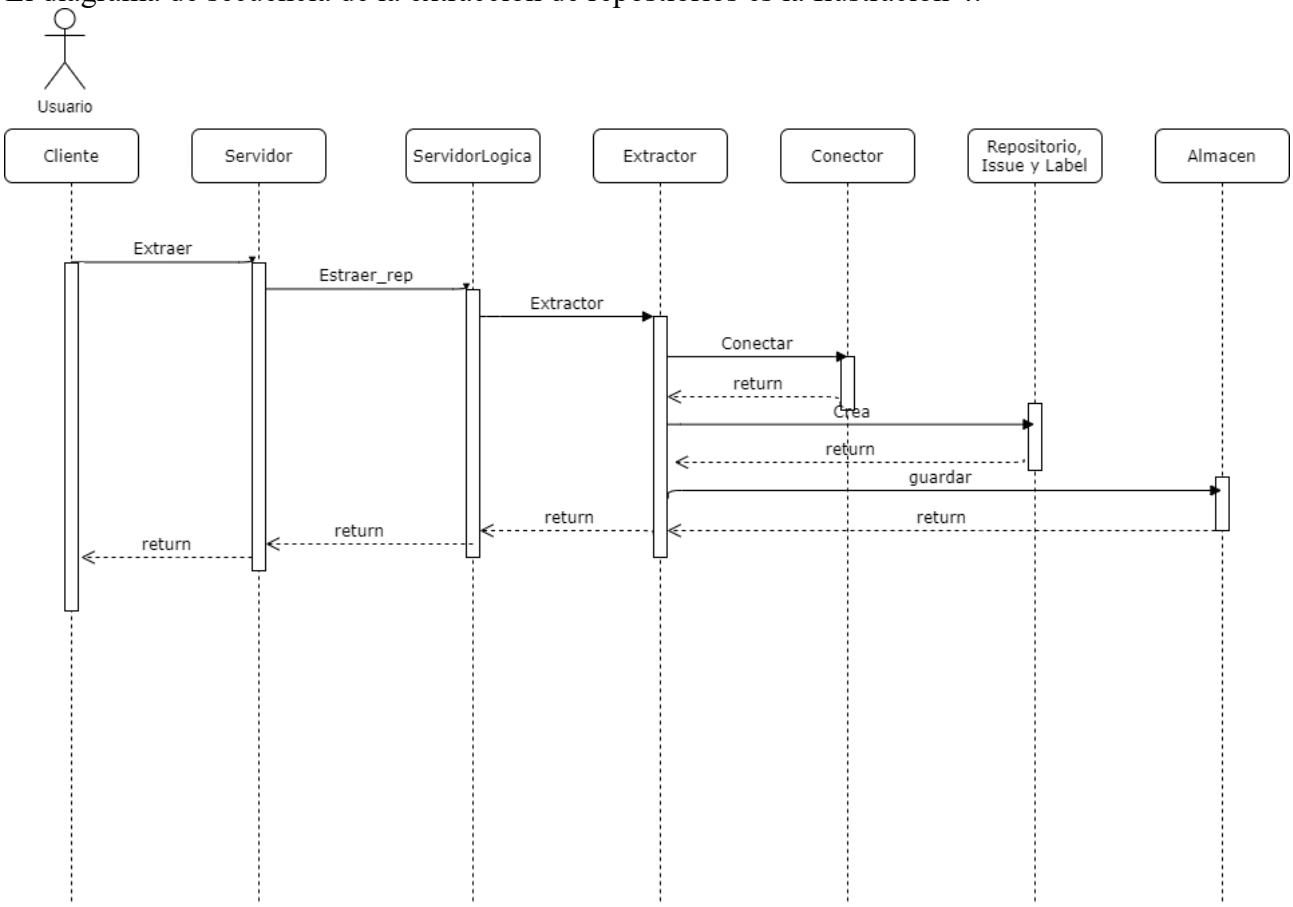

Ilustración 4: Diagrama de Secuencia de extracción

El diagrama de secuencia de la clasificación es Ilustración 5.

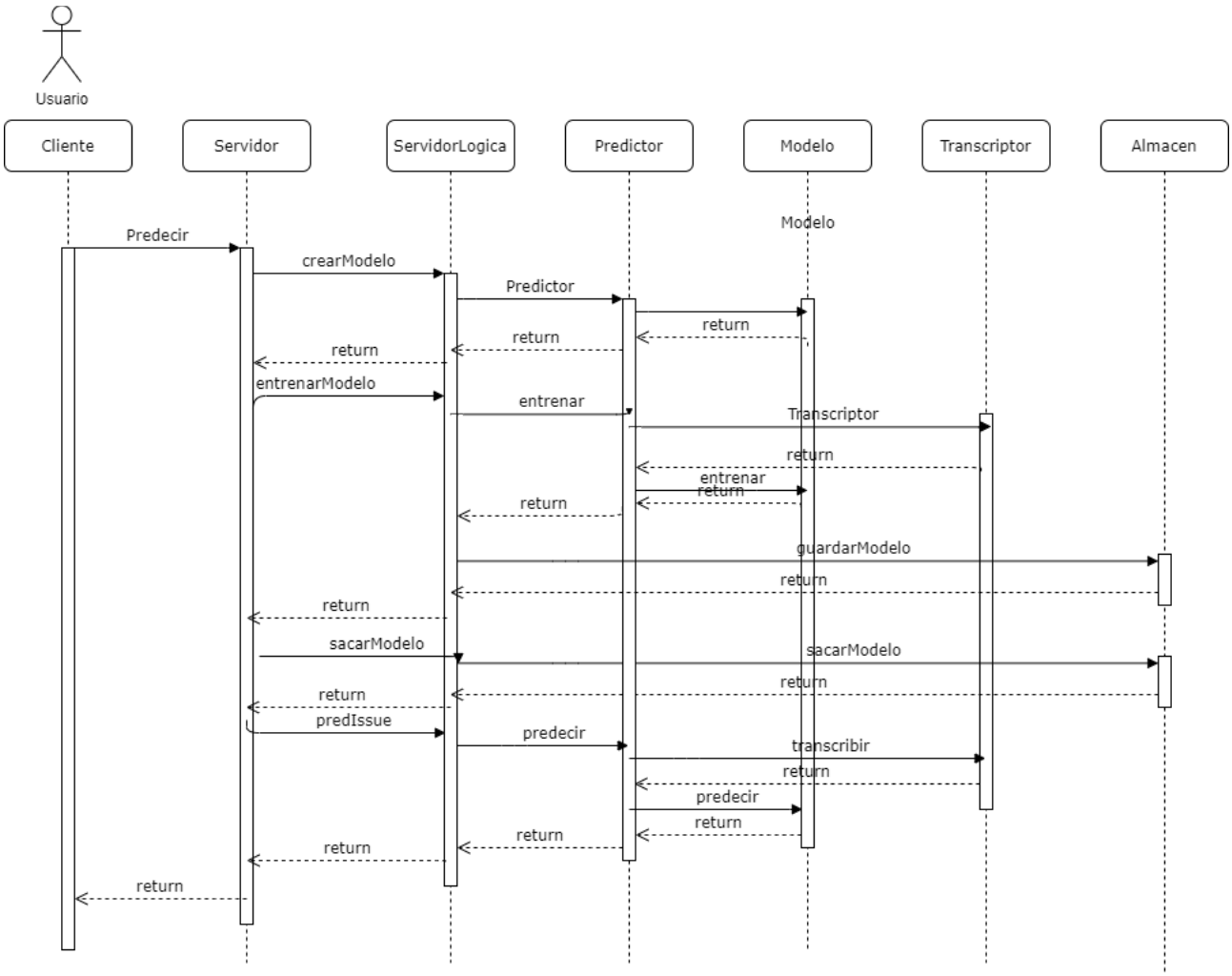

Ilustración 5: Diagrama de Secuencia Clasificación

. *Diseño arquitectónico*

En torno al diseño arquitectónico de la aplicación se ha utilizado una arquitectura cliente servidor en la cual la lógica del programa se encuentra en su totalidad en el servidor siendo el cliente una interfaz de comunicación con el usuario unicamente.

Apéndice D

IV - Documentación técnica de programación

. D.1. Introducción

En este apartado se explicará la documentación técnica de programación. Donde se explicará la estructura de directorios del proyecto, un manual para el programador sobre el entorno, las directivas de ejecución del proyecto y pruebas realizadas sobre el sistema.

. D.2. Estructura de directorios

La estructura del proyecto ha sido:

Carpeta raíz / carpeta dedicada a albergar el proyecto.

En ella podemos encontrar:

- lib
- src
- tst
- doc
- errores.txt

  Log de errores de la aplicación.
- requirements

  Archivo que contiene las dependencias de la aplicación.

Carpeta /lib:

Carpeta dedicada a archivos dedicados con la configuración del programa o usados por el mismo

Carpeta /src:

Carpeta dedicada al código fuente de la aplicación

En ella podemos encontrar los siguientes carpetas y archivos relevantes:

- formularios
- static
- templates
- Servidor.py

  Archivo que inicia el servidor y por tanto el programa.

Carpeta /src/formularios:

Carpeta dedicada a almacenar los formularios de Flask.

Carpeta /src/static solo está la carpeta css, en esta:

Carpeta dedicada a almacenar el archivo CSS

Carpeta /src/templates:

Carpeta dedicada a almacenar las páginas HTML.

Carpeta /tst:

Carpeta dedicada a las pruebas.

En ella se encuentra el archivo Prueba de desempeño.txt. Este archivo contiene las pruebas de desempeño realizadas sobre la aplicación.

Carpeta /doc:

Carpeta dedicada a la documentación del proyecto.

. D.3. Manual del programador

En este apartado detallaremos cuestiones del entorno de programación utilizado y describiremos cada uno de los archivos relevantes.

El IDE utilizado para el desarrollo de este proyecto ha sido Spyder, el cual viene dentro de la distribución de Anaconda. El proceso de instalación se encuentra en: https://www.spyder-ide.org/

Para desplegar la base de datos se ha utilizado Docker, posteriormente en el manual de instalación 25 se explica como configurarlo.

Para el MySQL Workbench se descarga el instalador en https://www.mysql.com/downloads/

Todos los archivos relacionados con el funcionamiento del programa se encuentran en la carpeta /src , en la carpeta /tst se encuentran las pruebas automáticas, que son los archivos que son terminados con 'con clases.py' y los archivos en los que se relaizan pruebas más básicas de como funcionan los elementos antes de integrarlos en la aplicación, que son el resto de las pruebas, a excepción de 'Prueba de Rendimiento.py' y los dos archivos de texto de Prueba de Desempeño, que son archivos relacionados con este tipo de pruebas.

Los errores que surjan a lo largo de la aplicación relacionados con la lógica del servidor irán a para al archivo /errores.txt.

Los archivos relacionados con la extracción de repositorios son: 'Token.txt 'de la carpeta /lib y de la carpeta /src 'Extractor.py', 'Conector.py', 'Issue.py', 'Label.py', y 'Repositorio.py', siendo estos tres últimos también relacionados con el almacenamietno.

Los archivos relacionados con el almacenamiento son: 'Conexión.txt' y 'Creacion de tablas.sql' de la carpeta /lib y 'Almacen.py' de la carpeta /src.

Los archivos relacionados con el entrenamiento y clasificación de los modelos: 'Stop.py' de la carpeta /lib y de la carpeta /src: 'Predictor', 'ModeloMultiClass', 'ModeloSingleClass', 'TranscriptorMultiClass' y 'TranscriptoSingleClass'.

Los archivos relacionados con la interacción con el usuario son todos de la carpeta /src y son: 'Servidor.py', 'ServidorLogica.py' y todos los de las tres subcarpetas de /src.

El inicio del programa se encuentra en 'Servido.py'

La descripción de los archivos:

En la carpeta /lib:

- Conexion.txt

  Archivo dedicado a guardar los parámetros de conexión con la base de datos
- Creación de tablas.sql

  Archivo dedicado a la creación de tablas de la base de datos.
- Stop.py

  Archivo dedicado a la descarga de las stopwords.
- Token.txt

  Archivo dedicadado a tener el token personal de acceso a Gitlab

En la carpeta /src:

- Almacen.py

  Clase dedicada a la comunicación con la base de datos
- Conector,py

  Clase encargada de conectarse con GitLab y crear el objeto.
- Extractor.py

  Clase encagada de la extrcción y adecuación de los datos del repostiorio
- Issue.py

  Clase que representa a una tarea.
- Label.py

  Clase que representa una etiqueta.
- ModeloMultiClass.py

  Clase dedicada a almacenar el modelo de predicción en modelos multiclase.
- ModeloSingleClass.py

  Clase dedicada a almacenar el modelo de predicción en modelos monoclase.
- Predictor.py

  Clase que ofrece una interfaz común entre los modelo monoclase y multiclase, el cual además se encarga de administrar todas las clases relacionadas con la predicción.
- Repositorio.py

  Clase que representa un repositorio
- Servidor.py

  Archivo que inicia el servidor y por tanto el programa. Clase que contiene las directivas del servidor Flask y gestiona las peticiones y envíos e interactua con servidor lógica.
- ServidorLogica.py

  Clase que contiene la lógica del programa y gestiona el mismo.

- TranscriptorMultiClass.py

  Clase encargada de hacer transformaciones en los datos de entrada y salida de los modelos multiclase para adaptarlos a la usabilidad necesaria.

- TranscriptorSingleClass.py

  Clase encargada de hacer transformaciones en los datos de entrada y salida de los modelos monoclase para adaptarlos a la usabilidad necesaria.

En la carpeta /src/formularios:

- FormularioExtracción.py

  Formulario encargado de obtener los datos de la pantalla de extracción.

- FormularioIssue.py

  Formulario encarado de obtener los datos relacionados con la issue a predecir.

- FormularioModelo.py

  Formulario encargado de intercambiar los datos relacionados con la recuperación de modelos de la base de datos.

- FormularioPredicción.py

  Formulario encargado de intercambiar los datos relacionados con el entrenamietno de un modelo.

En la carpeta /src/static solo está la carpeta css, en esta:

- base.css

  Archivo CSS de la interfaz gráfica.

En la carpeta /src/templates:

- base.html

  Página padre del que heredan los demás que tiene el titulo del trabajo, el boton de volver al inicio y la referencia al creador y los tutores de este trabajo.

- enlaceRoto.html

  Página que recibe el error de enlace roto

- error_c.html

  Página que recibe algunos errores

- extraccionCorrecta.html

  Página que certifica la extracción correcta.

- extraer.html

  Página que contiene el formulario de extracción.

- issues.html

  Página de predicción de issues.

- Macro_forms.html

  Macro para mostrar los campos de un formulario
- mainPage.html

  Página inicial del programa
- predecir.html

  Página de entrenamiento del modelo.
- recuperarModelo.html

  Página de carga de modelos almacenados

En la carpeta /tst:

- Prueba almacenamiento con clases.py

  Prueba de funcionalidad del almacenamiento.
- Prueba de almacenamiento.py

  Prueba de los mecanismos relacionados con el almacenamiento.
- Prueba de modelo con clases.py

  Prueba de funcionalidad de la predicción.
- Prueba de modelo.py

  Prueba de los mecanismos relacionados con la clasificación.
- Prueba de Rendimiento.py

  Prueba de desempeño de los modelos con determinadas opciones.
- Prueba de desempeño.txt

  Este archivo contiene las pruebas de desempeño realizadas sobre la aplicación.
- Prueba de Desempeño – Manual.txt

  Este archivo contiene las pruebas de desempeño realizadas sobre la aplicación hecho con una medida propia.
- Prueba de extracción con clases.py

  Prueba de funcionalidad del extracción.
- Prueba extracción.py

  Prueba de los mecanismos relacionados con la extracción..

. D.4. Compilación, instalación y ejecución del proyecto

Esto ya se explica posteriormente en el manual de instalación 25. Se ha de clonar el repositorios, tener una base de datos activa, correr la clase Servidor en un terminal y finalmente conectarse al puerto 5000 de la máquina con un navegador.

. D.5. Pruebas del sistema

- Se realizaron una pruebas de desempeño sobre el sistema con una configuración determinada: utilizando Stop words en inglés, oleyendo los comentarios, contando la spalabras con el método Count Vectorizer y teniendo en cuenta las tareas sin etiqueta y para repositorios variados como son:
- Proyecto de prueba del TFG. Project ID: 19766159.(https://gitlab.com/wgm1001/TFG-tst)
- Mailman Core. Project ID: 183616 (https://gitlab.com/mailman/mailman)
- Foundry VTT 5th Edition. Project ID: 8860457 (https://gitlab.com/foundrynet/dnd5e)
- Commento. Project ID: 6094330 (https://gitlab.com/commento/commento)
- inkscape. Project ID: 3472737 (https://gitlab.com/inkscape/inkscape)
- Client. Project ID: 36189 (https://gitlab.com/fdroid/fdroidclient)

Repositorio	Algoritmo	Tiempo entrenamiento	Tiempo de predicción	Jaccard accuracy	F1 score	Precision Score	Recall score
19766159	Multinomial NB	0.06101s	0.00099s	0.8571429	0.9230769	1.0000000	0.8571429
19766159	SVM	0.06000s	0.00100s	0.8571429	0.9230769	1.0000000	0.8571429
19766159	KNN	0.05799s	0.00301s	0.3000000	0.4615385	0.5000000	0.4285714
19766159	RandomForest	0.12653s	0.04200s	0.8571429	0.9230769	1.0000000	0.1428571
19766159	OneVsRest	0.05999s	0.00401s	0.4000000	0.5714286	0.5714286	0.5714286
183616	Multinomial NB	1.80753s	0.82023s	0.5592287	0.7173145	0.8141711	0.6410526
183616	SVM	34.06570s	11.02722s	0.3584000	0.5276796	0.5989305	0.4715789
183616	KNN	2.40453s	13.20249s	0.4268908	0.5983510	0.6791444	0.5347368
183616	RandomForest	10.25135s	5.96122s	0.7873684	0.8810365	1.0000000	0.7873684
183616	OneVsRest	3.48602s	5.19308s	0.0044030	0.0087674	0.0058060	0.0178947
8860457	Multinomial NB	0.36000s	0.16551s	0.7156208	0.8342412	0.8772504	0.7952522
8860457	SVM	6.01458s	3.00607s	0.4569161	0.6272374	0.6595745	0.5979228
8860457	KNN	0.71601s	4.47013s	0.3787554	0.5494163	0.5777414	0.5237389
8860457	RandomForest	2.38209s	4.79208s	0.8814056	0.9369650	0.9852700	0.8931751
8860457	OneVsRest	0.67999s	1.89702s	0.0073378	0.0145688	0.0090645	0.0370920
6094330	Multinomial NB	0.42999s	0.14500s	0.5000000	0.6666667	0.9032258	0.5283019
6094330	SVM	5.51700s	1.61300s	0.3237822	0.4891775	0.6627566	0.3876501
6094330	KNN	0.71901s	2.40900s	0.3371925	0.5043290	0.6832845	0.3996569
6094330	RandomForest	3.60863s	2.70054s	0.5849057	0.7380952	1.0000000	0.5849057
6094330	OneVsRest	0.56099s	1.54508s	0.0121304	0.0239700	0.0212766	0.0274443
3472737	Multinomial NB	236.98142s	9.65301s	0.2732581	0.4292265	0.7601516	0.2990415
3472737	SVM	2200.81975s	385.31816s	0.1521663	0.2641394	0.4677856	0.1840256
3472737	KNN	53.78648s	544.51506s	0.2537371	0.4047692	0.7168381	0.2820021
3472737	RandomForest	243.47863s	15.01872s	0.3933972	0.5646591	1.0000000	0.3933972
3472737	OneVsRest	124.07131s	36.62875s	0.0347677	0.0671990	0.1148301	0.0474973
36189	Multinomial NB	40.90787s	2.98210s	0.4297970	0.6012000	0.7447968	0.5040241
36189	SVM	449.79903s	128.16888s	0.3376137	0.5048000	0.6253717	0.4232059
36189	KNN	17.58970s	175.75340s	0.3777900	0.5484000	0.6793855	0.4597586

Repositorio	Algoritmo	Tiempo entrenamiento	Tiempo de predicción	Jaccard accuracy	F1 score	Precision Score	Recall score
36189	RandomForest	53.68195s	15.48990s	0.6761649	0.8068000	0.9995045	0.6763917
36189	OneVsRest	51.04547s	21.96582s	0.0143217	0.0282390	0.0179086	0.0667337

Tabla 5: Pruebas de desempeño

De estos resultados podemos concluir que depende mucho de las características del programa, y que en repositorios muy grande suele tener malos resultados

Apéndice E

V - Documentación de usuario

. E.1. Introducción

En este apéndice se detallará el manual de instalación y el manual de usuario además de de los requisitos de usuario

. E.2. Requisitos de usuarios

Los requisitos que ha de cumplir el usuario para poder utilizar la aplicación es tenerla correctamente instalada, tener la base de datos y el servidor encendidos y tener un navegador con el que conectarse. En caso de no ejecutarlo en local ha de tener el puerto 5000 abierto.

. E.3. Instalación

Para ejecutar este programa será necesario instalar los siguientes elementos:

Para empezar habrá que instalar Python, en la versión 3.8.0. La página de descarga es: https://www.python.org/downloads/ y también se proveerá de manual de instalación en ella (https://wiki.python.org/moin/BeginnersGuide/Download).

Lo siguiente que será necesario es clonar o descargarse el repositorio desde Github, tras lo cual habrá que abrir un interprete de comandos en la carpeta del proyecto, o bien ir hasta ella, y ejecutar el siguiente comando:

```
python -m pip install -r requirements.txt
```

Será necesario también correr los siguientes comandos en una terminal en las localizaciones marcadas:

En la localización donde se tenga instalado Python 3.8 el comando:

```
Install\ Certificates.command
```

En la localización relativa al programa /lib: `python Stop.py`

También es necesario instalar una base de datos MySql, para esto hay varias alternativas:

Se puede instalar Docker y descargar una imagen de mysql. Para instalar Docker aquí está el instalador para Windows https://hub.docker.com/editions/community/docker-ce-desktop-windows y aquí la guía de instalación https://docs.docker.com/docker-for-windows/install/ y para la descarga y puesta en marcha de la imagen de MySQL https://hub.docker.com/_/mysql

En caso de querer instalar la base de datos MySQL aquí está el link de la guía de instalación https://dev.mysql.com/doc/mysql-installation-excerpt/5.7/en/

Una vez instalado se deberá colocar en el archivo 'Conexion.txt' un usuario y contraseña, de una cuenta con permisos para realizar inserciones y consultas sobre la base de datos, en los apartados user y passwd, entre comillas simples. En este archivo también se debe rellenar los demás campos,

el host habiendola instalado de forma local no hay que alterarlo, en caso de no hacerlo insertar la dirección IP de la máquina a la que se haya que conectar. El puerto será en el que se estén gestionando las peticiones de la base de datos. El parámetro db no hay que cambiarlo ya que representa el esquema al que estamos accediendo.

Para la creación de tablas hay que correr el archivo 'Creacion de tablas.sql' en tu base de datos.

Finalmente se deberá generar un token de acceso en https://gitlab.com/profile/personal_access_tokens (Para lo cual se necesita tener cuenta en Gitlab) y meter ese token en el archivo Token.txt de la carpeta lib.

Ahora para correr el programa se necesita tener activa la base de datos, desde la carpeta src correr el archivo 'Servidor.py' con el comando:

```
python Servidor.py.
```

Para conectarse al programa se utilizará un navegador abrir localhost:5000/ (en caso de no estar ejecutando en local cambiar localhost por la dirección del dispositivo)

. *E.4. Manual del usuario*

. Inicio

Para iniciar se ha de tener encendida la base de datos y configurados los parámetros de los archivos ‘Conexión.txt’ y ‘Token.txt’ tal y como viene indicado en el manual de instalación. Para iniciar el servidor es necesario correr el archivo ‘Servidor.py’ con el comando (desde la carpeta src):

Python Servidor.py

Finalmente para conectarse abrir un navegador y escribir en la barra de direcciones: http://localhost:5000/

. Extracción de repositorios

Para la extracción de repositorios desde la pantalla inicial hay que pulsar el botón extraer.

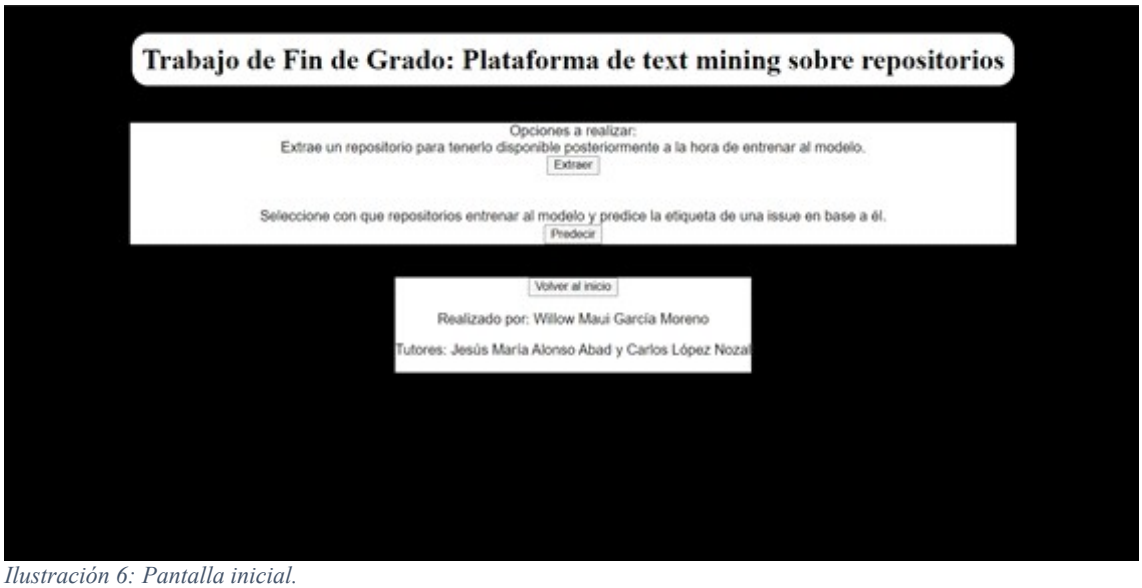

Ilustración 6: Pantalla inicial.

Desde la pantalla de extraer en el recuadro de ‘Url’ se ha de colocar la dirección de la pantalla principal del proyecto en Gitlab.

En caso de querer otro token de acceso a Gitlab distinto al provisto en el archivo ’Token.txt’ este ha de colocarse en el recuadro ‘Token privado’.

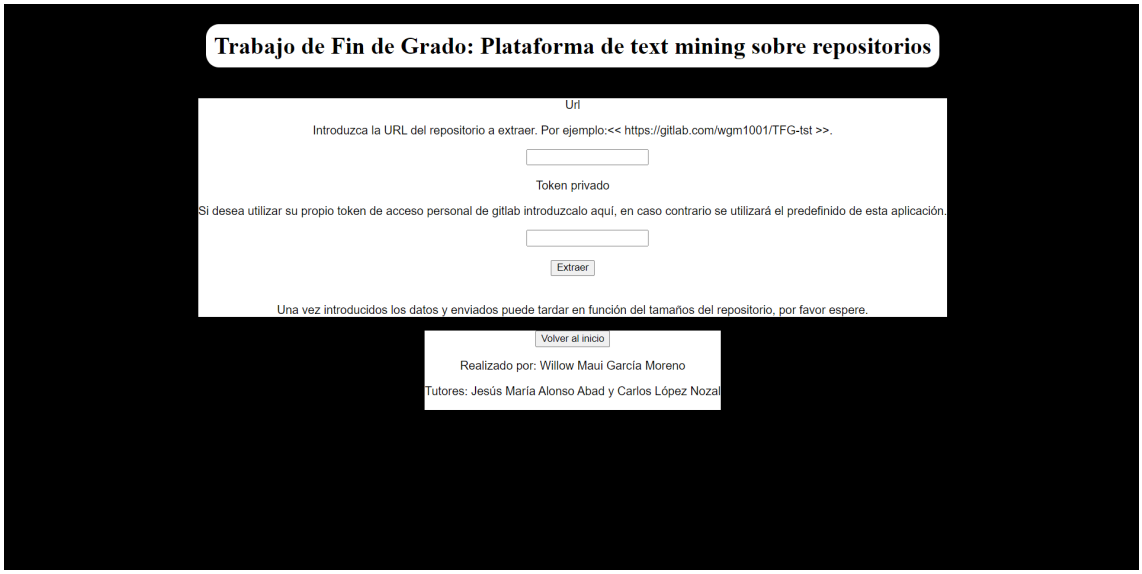

Ilustración 7: Pantalla de extracción.

Una vez introducidos los datos, si todo va bien, una vez extraído el repositorio se cambiará a la pantalla de extracción correcta.

Ilustración 8:Pantalla de extracción correcta.

. Predicción de etiquetas

Para la predicción de etiquetas se requiere haber extraído algún repositorio, ya que se debe entrenar el clasificador con al menos un repositorio.

Para predecir desde la pantalla de inicio hay que dar al botón predecir, lo cual nos llevará a la pantalla de selección de modelo, en la cual podremos rellenar los distintos parámetros para entrenar un clasificador con unas características concretas o podremos, en caso de haberlo entrenado en otro momento dar al botón de ‘Cargar modelo’ lo que nos llevará a la pantalla de selección de un modelo ya entrenado. Una vez Introducidos los parámetros al dar al botón ‘Entrenar’ se iniciará el entrenamiento del modelo y una vez finalizado se pasará a la pantalla de predicción de etiquetas además de guardar el modelo entrenado en la base de datos. Los algoritmos MultinomialNB, SVM, KNN y Random Forest solo son capaces de predecir una etiqueta mientras que no se genere un modelo por etiqueta.

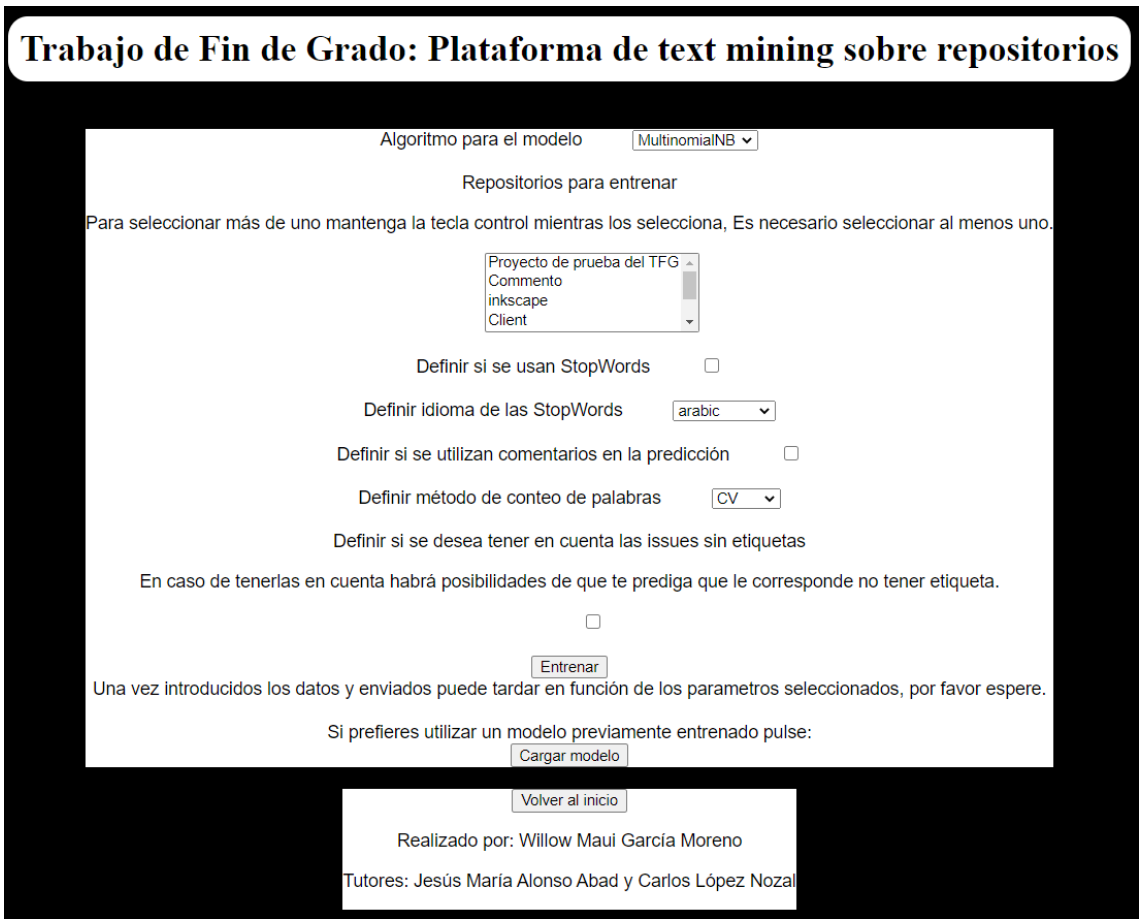

Ilustración 9: Pantalla de selección de modelo.

Desde la pantalla de selección de un modelo ya entrenado se puede escoger entre los modelos que ya se han entrenado para la predicción de las issues, pasando a la pantalla de predicción de etiquetas.

Ilustración 10:Pantalla de selección de modelo previamente entrenado.

En la pantalla de predicción de etiquetas se podrán introducir los parámetros requeridos para predecir la issue y dar al botón 'Predecir' para que nos muestre las etiquetas predichas una vez. También se puede desplegar la pestaña de 'Detalles del modelo' para ver los parámetros de entrenamiento del modelo que se está utilizando.

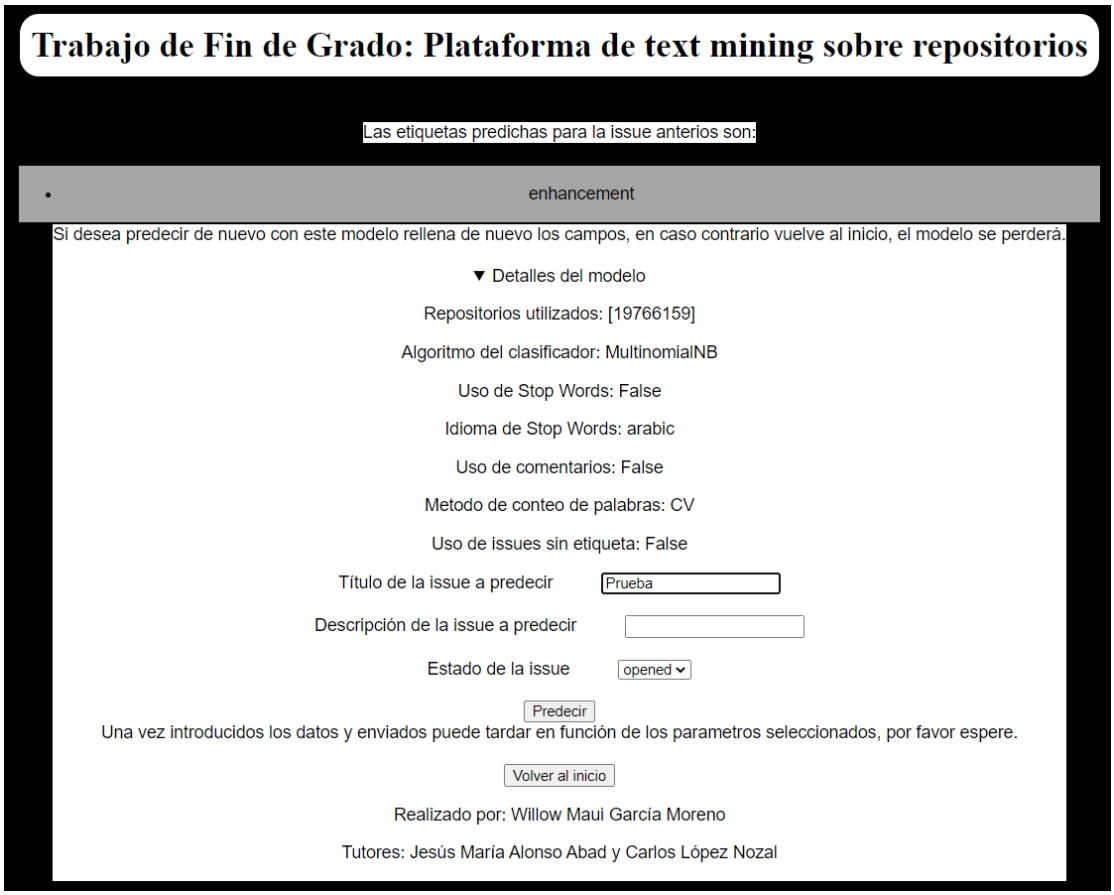

Ilustración 11: Pantalla de predicción de etiquetas.

. Pantalla de error

En caso de algún error en la introducción de los datos saltará la pantalla de error, desde la cual podrás volver a la pantalla de inicio y empezar de nuevo.

Ilustración 12: Pantalla error