API para entrenamiento de redes neuronales profundas evolucionadas a través de algoritmos genéticos

Trabajo Final de Grado
Grado en Ingeniería Informática
Especialidad Ingeniería de Software
Universidad del País Vasco

Autor: Leroy Deniz Pedreira
Tutor: Imanol Usandizaga Lombana

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

A Cristina.

“Hay quienes creen que el destino descansa en las rodillas de los dioses, pero la verdad es que trabaja, como un desafío candente, sobre la conciencia de los hombres.”

EDUARDO GALEANO
Las venas abiertas de América Latina, 1971.

Control de versiones

Versión	Fecha	Autor	Descripción
1.0.0	01/04/21	LD	Definición de la estructura de la memoria del TFG
1.0.1	03/04/21	LD	Definición del alcance y objetivos
1.0.2	04/04/21	LD	Definición de resumen ejecutivo, abstract, cronograma macro de deadlines y planificación general
1.0.3	07/04/21	LD	Desglose de tareas y diagrama de Gantt
1.1.0	17/04/21	LD	Corrección del resumen ejecutivo y abstract. Inscripción del TFG en GAUR, ajustes en la planificación, adquisición de información bibliográfica.
1.1.1	18/04/21	LD	Relevamiento de riesgos y finalización de la planificación
1.1.2	24/04/21	LD	Definición de los casos de uso
1.2.0	25/06/21	LD	Reestructura de la planificación.
1.2.1	26/06/21	LD	Análisis y elección de herramientas de Frontend. Desarrollo de la interfaz gráfica. Módulo de vinculación entre GUI y API.
1.2.2	27/06/21	LD	Actualización del glosario de términos y tecnologías, optimizaciones en el proceso de consulta a las operaciones del webservice en el frontend.
1.2.3	28/06/21	LD	Actualización de los casos de uso. Mejoras en la interfaz gráfica. Actualización de las funcionalidades de entrenamiento, clasificación y validación por medio de condiciones de radio buttons.
1.2.4	29/06/21	LD	Actualización de la planificación
1.2.5	01/07/21	LD	Nuevas tecnologías utilizadas registradas. Tratamiento de los módulos de comunicación entre la API con el servidor.
1.2.6	02/07/21	LD	Creación de diagramas de componentes y diagramas de secuencia de sistema.
1.2.7	03/07/21	LD	Se agrega la hipótesis, análisis de antecedentes y comienzo del marco teórico.
1.2.8	04/07/21	LD	Continuación del marco teórico y antecedentes. Actualización del esquema de la memoria.
1.2.9	05/07/21	LD	Finalización del marco teórico. Actualización de la gestión del proyecto.
1.2.10	06/07/21	LD	Actualización de las tecnologías utilizadas. Actualización de los diagramas de secuencia ajustados a los nuevos tipos de datos vinculados a la API. Actualización de la descripción del proyecto.
1.3.0	07/07/21	LD	Diseño de la solución de la API, definición de clases, criterios, fases y operadores de Algoritmos Genéticos y Redes Neuronales. Actualización de la bibliografía y corrección de estilo del marco teórico.

1.3.1	08/07/21	LD	Actualización de la bibliografía, marco teórico de Frontend. Creación de tabla de imágenes. Documentación del proceso de preparación de entorno de la API.
1.3.2	09/07/21	LD	Descripción de la configuración del entorno de producción. Actualización de las tecnologías utilizadas y las capturas de pantalla de la interfaz gráfica de la aplicación.
1.3.4	10/07/21	LD	Primera parte de la instalación del servidor de producción y su documentación. Marco teórico de la evolución de las redes neuronales a través de algoritmos genéticos. Registro de antecedentes de proyectos anteriores.
1.3.5	11/07/21	LD	Descripción de la instalación y configuración del entorno de producción. Añadido el marco teórico sobre diseño de RNAs sobre AGs.
1.4.0	12/07/21	LD	Implementación de la estructura de las funciones de recepción y envío desde la API. Vinculación de la GUI en React a los webservices en la API utilizando AJAX.
1.4.1	13/07/21	LD	Desarrollo de las validaciones de los datos de entrada y pre procesado de datos.
1.4.2	14/07/21	LD	Replanificación. Eliminación del paquete LLAA de la planificación inicial. Corrección de errores de la API cuando recibe archivos demasiado grandes.
1.4.3	15/07/21	LD	Se agregan fuentes a las imágenes y pseudocódigos.
1.4.4	17/07/21	LD	Actualización de la estructura de la memoria. Ajustes en el proceso de implementación de la interfaz de comunicación de la API.
1.4.5	19/07/21	LD	Documentación e implementación del Exploratory Data Analysis con Flask y Pandas. Definición de política de preprocessing e implementación de la función.
1.4.6	20/07/21	LD	Corrección de preprocessing.
1.5.0	22/07/21	LD	Actualización del plan de calidad y definición de la estrategia de testing a aplicar.
1.5.1	23/07/21	LD	Desarrollo del Core y descripción del contenido. Se añaden citas al comienzo de cada capítulo. Se referencia algo más sobre algoritmos genéticos en el marco teórico.
1.6.0	24/07/21	LD	Herramientas de testing y estrategia a implementar en la automatización.
1.6.1	25/07/21	LD	Implementación de la red neuronal y documentación del proceso. Actualización de las tareas realizadas y ajuste en las dedicaciones. Primer diseño del análisis de la calidad a desarrollar.
1.6.2	26/07/21	LD	Desarrollo y documentación de la estructura del algoritmo genético, incorporación a la red neuronal y evaluación del gen. Inicio del operador de Selección.
1.6.3	27/07/21	LD	Desarrollo y documentación de los operadores de selección y mutación.
1.6.4	28/07/21	LD	Se añaden comentarios al código y se agregan las funciones al apartado de Implementación del Algoritmo Genético. Desarrollo y documentación de los operadores de selección y reproducción. Descripción de vinculación de la API con flask y devolución del webservice.

1.7.0	29/07/21	LD	Documentación de los cambios y ajustes en las vistas. Puesta en producción de las funciones y ajustes de los servicios web contra las vistas.
1.7.1	30/07/21	LD	Continuación de la documentación del algoritmo genético. Análisis de conjuntos de datos de ejemplo y casos límite. Síntesis de las conclusiones. Generación de casos de prueba y finalización de los casos de uso Clasificar y Evaluar red neuronal.
1.7.2	31/07/21	LD	Actualización del apartado de conclusiones y mejora de la documentación del algoritmo genético. Mejora de la usabilidad de las vistas de la aplicación.
1.7.3	01/08/21	LD	Finalización de ajustes resultantes del análisis de calidad. Actualización de seguimiento y control.
1.7.4	02/08/21	LD	Maquetación del documento.

Tabla de contenidos

Resumen ejecutivo

Los algoritmos de clasificación supervisada resuelven un tipo de problemas específicos donde se cuenta con una serie de datos previamente clasificados. Para poder decidir la clase de una muestra, se tiene en cuenta toda la información que se pueda extraer del conjunto de datos inicial, cuya clase es conocida por el clasificador.

En este trabajo se implementa una aplicación web, que permite al usuario interactuar a través de una API (Application Programming Interface), capaz de entrenar y aproximar muy bien un óptimo global de una red neuronal profunda mediante algoritmos genéticos, con las funcionalidades de entrenamiento, evaluación y clasificación.

La aplicación es construida sobre una arquitectura de tres capas, contando con una interfaz web diseñada en ReactJS, que permitirá al usuario interactuar con las funcionalidades del software de clasificación.

Se aborda el diseño del Core utilizando técnicas de clasificación supervisada bajo entrenamiento de redes neuronales profundas, optimizándolas a través de backpropagation y evolucionándolas con algoritmos genéticos. La API es construida en Python con Flask a nivel de Controlador y PyTorch y DEAP como frameworks en el Core. Este último será el responsable de entrenar la red neuronal y hacerla evolucionar hasta lograr una aproximación muy cercana al óptimo global (Castillo Valdivieso et al., 2000). El uso de metaheurísticas permite a la red modificar sus pesos y crecer o reducir, tanto en número de nodos como de capas de manera dinámica, en función de lo que ella misma considere necesaria para su propio entrenamiento.

Palabras clave: servicios web, react framework, redes neuronales, algoritmos genéticos, algoritmos evolutivos, problemas de optimización, aprendizaje automático, minería de datos.

Abstract

Supervised classification algorithms solve a specific type of problem where a series of previously classified data is available. In order to decide the classification of a sample, all the information that can be extracted from an initial data set is taken into account, with data whose class is known by the classifier.

In this work, a web application is implemented that allows the user to interact through an API (Application Programming Interface), that is able to train and approximate accurately a global optimal from a deep neural network by means of genetic algorithms, able to traint, evaluate and classify.

The application is built on a three-layer architecture, with a web interface designed in ReactsJS, that will allow the user to interact with the functionalities provided by the classification software.

The design of the Core is approached using supervised classification techniques under training of deep neural networks by means of backpropagation and making them evolve with genetic algorithms. The API is built in Python making use of PyTorch and DEAP as frameworks in the Core level and Flask in the Controller level. This Core will be responsible for training the neural network and making it evolve until it reaches a very close approximation as far as the global optimal is concerned (Castillo Valdivieso et al., 2000). The use of genetic algorithms enables the network to modify its weights and grow not only in the number of nodes, but also in the number of layers in a dynamic way, based on what it considers necessary for its own training.

Keywords: web services, react framework, neural network, genetic algorithms, evolutives algorithms, optimization problems, machine learning, data mining.

Capítulo I

Planificación del Proyecto

Planificación del proyecto

"Si tuviese sólo una hora para salvar el mundo, dedicaría cincuenta y cinco minutos a definir bien el problema."

ALBERT EINSTEIN

Alcance

El alcance del proyecto GANN incluye el trabajo necesario para producir y entregar un sitio web y una API, cuya funcionalidad implica resolver problemas de clasificación supervisada, entrenando redes neuronales profundas evolucionadas a través de algoritmos genéticos.

La aplicación se construirá con una arquitectura de tres capas, sus responsabilidades estarán previamente definidas y la comunicación entre GUI y el core será a través de una API. El producto final del proyecto será un sitio web hecho en React, una API en PyTorch, DEAP y Flask, y la documentación que especifique el diseño y construcción, así como una referencia de su uso.

La herramienta estará disponible bajo licencia AGPLv3, ya que permite la compatibilidad con algunas de las tecnologías aquí utilizadas.

Objetivos

Objetivo general

Desarrollar un software que resuelva problemas de clasificación supervisada, entrenando redes neuronales profundas evolucionándolas a través de algoritmos genéticos.

Objetivos específicos

1. Definir las características de una red neuronal profunda evolutiva optimizada en función del dataset etiquetado cargado por el usuario.
2. Implementación del módulo de la API utilizando Pytorch, Numpy y otros frameworks .
3. Implementación de la interfaz de comunicación de la API utilizando Flask.
4. Implementar una política de tratamiento de datos incompletos a través de Pandas.
5. Implementación de la interfaz gráfica de comunicación con el usuario con React.

Hipótesis

La hipótesis inicial del proyecto busca demostrar que es posible alcanzar la creación automática y eficiente de una red neuronal profunda, evolucionando de una estructura mínima y ésta es capaz de adaptarse a un problema, independientemente de su contexto utilizando algoritmos evolutivos.

Deadlines de entregas

Fecha	Descripción
25/06/21	Planificación actualizada
28/06/21	Prototipo GUI
30/06/21	Prototipo comunicación GUI - API / API - GUI
15/07/21	Marco teórico y antecedentes
25/07/21	Prototipo API
24/07/21	Prototipo paralelizado
25/07/21	Resultados de análisis de calidad
27/08/21	Entrega de la memoria

EDT/WBS

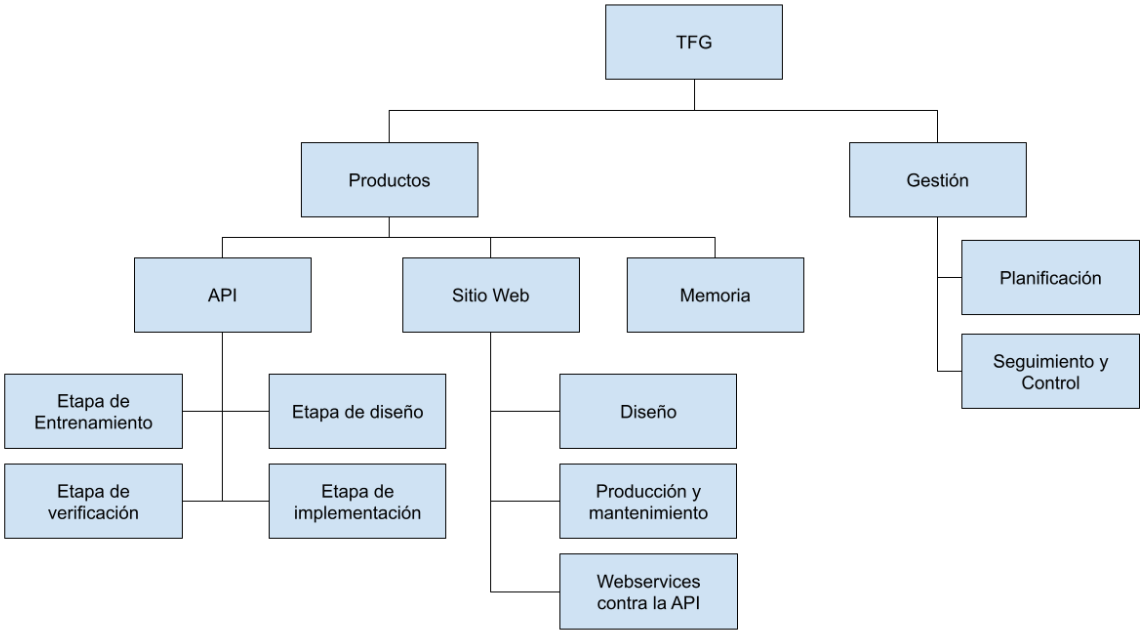


Figura 1: Estructura de desglose de trabajo.

Estimación inicial por paquete

La distribución de las 300 horas establecidas para el trabajo final de grado, se prevé una distribución horaria en función de los paquetes de trabajo de la forma en la que se establece debajo.

Paquete	Dedicación	Fechas
Planificación	20 horas	01/04/21 - 20/04/21
Sitio web	31 horas	26/06/21 - 01/07/21
API	103 horas	28/06/21 - 25/07/21
Memoria	58 horas	21/04/21 - 27/07/21
Seguimiento y control	48 horas	21/04/21 - 27/07/21
Lecciones aprendidas	10 horas	26/07/21 - 27/07/21
Horas de contención (10%)	30 horas	--
Total	**300 horas**	**01/04/21 - 27/07/21**

Marco estructural

Supuestos

A la hora de diseñar el sitio web que servirá de front end para el uso de las soluciones que aquí se presentan, se elegirá entre aquellas tecnologías web que, o bien se conozcan con suficiente soltura, o su aprendizaje no represente un desafío a la hora del diseño de la capa de presentación y comprometa las horas asignadas.

Aunque la aplicación debería ser capaz de procesar cualquier dataset, aquí se definirá una política de cuál es el formato que aceptará, con el fin de facilitar la implementación y todos los casos de uso derivados que pudieran haber.

Exclusiones

En función del tiempo que se tiene para el desarrollo de este proyecto, se excluye un estudio sobre la optimización de la plataforma que mejor rendimiento tenga, utilizando un servidor Debian 10 como entorno de producción de la aplicación.

Además, aunque la propuesta plantea la paralelización de los procesos de clasificación y procesamiento de datos, no se analizará la mejora del tiempo utilizando herramientas alternativas.

Todas estas exclusiones pueden servir como insumo para mejoras a futuro si se quisiera mejorar la aplicación, de cara a alcanzar un uso práctico o masivo.

Privacidad de los datos

Los DataSets utilizados para en el marco de este proyecto, tanto para las pruebas en la etapa de desarrollo como en la presentación de resultados, son tomados de UCI (https://archive.ics.uci.edu/), siendo esta una plataforma de recursos abiertos para Machine Learning.

Además, el desarrollo de esta herramienta no prevé el guardado de información en ningún soporte.

Desglose de tareas

Nombre de tarea	Horas	Comienzo	Fin
Planificación	**20**	**01-04-21**	**20-04-21**
Definición del tema	1	01-04-21	01-04-21
Definición de alcance	1	02-04-21	02-04-21
Definición de objetivos	3	02-04-21	02-04-21
Definición de resumen ejecutivo y abstract	2	02-04-21	02-04-21
Definición de cronograma macro y deadlines	1	05-04-21	05-04-21
Desarrollo de planificación general	2	06-04-21	08-04-21
Desglose de tareas y diagrama de Gantt	5	09-04-21	19-04-21
Entrega planificación inicial	5	20-04-21	20-04-21
Sitio web	**31**	**26-06-21**	**01-07-21**
Diseño, maquetación y contenido	**17**	**26-06-21**	**28-06-21**
Análisis de herramientas	1	26-06-21	26-06-21
Aprendizaje sintaxis de ReactJS	8	26-06-21	26-06-21
Preparar entorno de trabajo	1	27-06-21	27-06-21
Diseño de componentes y maquetación	7	27-06-21	28-06-21
Interfaz de comunicación webservice	**8**	**28-06-21**	**30-06-21**
Implementación módulo de vinculación	6	28-06-21	30-06-21
Tratamiento de resultados	2	30-06-21	30-06-21
Funcionalidades complementarias	**6**	**30-06-21**	**01-07-21**
Exportar XML	2	30-06-21	01-07-21
Exportar JSON	2	30-06-21	01-07-21
Exportar CSV	2	30-06-21	01-07-21
API	**103**	**28-06-21**	**25-07-21**
Etapa de Diseño	**16**	**28-06-21**	**01-07-21**
Modelos de Casos de Uso	3	28-06-21	27-06-21
Modelo de Dominio	3	30-06-21	30-06-21
Flujo de eventos	5	29-06-21	29-06-21
Diagramas de Secuencia de Sistema	5	01-07-21	01-07-21
Etapa de Implementación	**57**	**02-07-21**	**24-07-21**
Implementación del módulo de algoritmo genético	25	02-07-21	12-07-21

Implementación del módulo de red neuronal	25	13-07-21	23-07-21
Paralelización de procesamiento	7	11-07-21	24-07-21
Etapa de Entrenamiento	**15**	**11-07-21**	**24-07-21**
Definición del módulo de entrenamiento en API	6	21-07-21	23-07-21
Definición del módulo de clasificación en Controller	7	13-07-21	23-07-21
Establecimiento de JSON de retorno	2	11-07-21	24-07-21
Etapa de Verificación	**15**	**12-07-21**	**25-07-21**
Testeo del módulo de redes neuronales	5	12-07-21	13-07-21
Testeo del módulo de algoritmos genéticos	5	23-07-21	24-07-21
Análisis de DataSets (workbench)	5	24-07-21	25-07-21
Memoria	**58**	**21-04-21**	**27-07-21**
Definición de estructura y contenido	3	21-04-21	29-04-21
Documentación de tecnologías utilizadas	6	28-06-21	30-06-21
Búsqueda de antecedentes	4	02-07-21	03-07-21
Contextualización y marco teórico	15	01-07-21	15-07-21
Descripción del desarrollo Frontend	8	28-06-21	30-06-21
Descripción del desarrollo API	8	01-07-21	24-07-21
Estructura de datos de entrada	3	30-06-21	30-06-21

Metodología de trabajo

En función de la distribución seccionada de paquetes de trabajo y las etapas de cada uno, las fechas de cada tarea define la secuencia en la que se puede trabajar en cada una de ellas. Para esta primera parte, se opta por una metodología de desarrollo ágil, con miras de establecer en el trabajo futuro una de tipo iterativa incremental, prestando especial atención en la planificación, en el desglose de trabajo, alcanzando así un nivel de granularidad lo más fino posible en las tareas.

Riesgos

Métricas para la cuantificación y cualificación de riesgos

Probabilidad	Valor numérico	Impacto	Valor numérico	Tipo de riesgo	Probabilidad x impacto
Nada probable	0.10	Muy bajo	0.05	Muy alto	Mayor a 0.50
Poco probable	0.30	Bajo	0.10	Alto	Menor a 0.50
Probable	0.50	Moderado	0.20	Moderado	Menor a 0.30
Muy probable	0.70	Alto	0.40	Bajo	Menor a 0.10
Casi probable	0.90	Muy alto	0.80	Muy bajo	Menor a 0.05

Matriz de identificación de riesgos

Cod.	Tipo	Descripción / Causa raíz	Tipo resp	Estim. prob.	Objet. afect	Estim. de impacto	Prob x imp	Tipo riesgo (cualit)
RI001	I	Planificación optimista.	EV	0.3	AL			M
					TI	0.4	0.12	
					CO	0.2	0.06	
					CA			
					∑ prob*impacto		0.18	
RI002	I	Tareas no contempladas en el desglose inicial.	MI	0.3	AL	0.05	0.015	M
					TI	0.4	0.12	
					CO	0.2	0.06	
					CA			
					∑ prob*impacto		0.195	
RI003	I	Los requisitos no han sido relevados correctamente, provocando una redimensión necesaria del alcance.	TR	0.1	AL	0.8	0.08	B
					TI	0.2	0.02	
					CO			
					CA			
					∑ prob*impacto		0.10	
RI004	I	El diseño inicial presenta errores graves y es necesario un rediseño e implementación.	EV	0.3	AL	0.8	0.24	A
					TI	0.4	0.12	
					CO	0.4	0.12	
					CA			
					∑ prob*impacto		0.48	

RI005	I	La falta de motivación repercute en el nivel de productividad..	AC	0.3	AL			A
					TI	0.4	0.12	
					CO			
					CA	0.8	0.24	
					∑ prob*impacto		0.36	
RE001	E	Las tecnologías utilizadas para la implementación de los módulos no se integran de la manera esperada.	EV	0.5	AL			A
					TI	0.4	0.2	
					CO			
					CA	0.4	0.2	
					∑ prob*impacto		0.4	
RE002	E	Las actualizaciones de las herramientas provocan una curva de aprendizaje más acentuada.	AC	0.5	AL	0.8	0.4	MA
					TI	0.8	0.4	
					CO			
					CA			
					∑ prob*impacto		0.8	
RE003	E	Las herramientas de verificación muestran que no se alcanza la línea base de la calidad.	ME	0.3	AL			A
					TI	0.4	0.12	
					CO	0.2	0.06	
					CA	0.8	0.24	
					∑ prob*impacto		0.42	

Tipo de riesgo: I - interno, E - Externo.
Objetivo: AL - Alcance, TI - Tiempo, CO - Costo, CA - Calidad
Tipo de respuesta: MI - Mitigar, EV - Evitar, TR - Transferir, EX - Explotar, CO - Compartir, ME - Mejorar, AC - Aceptar
Tipo de riesgo cualitativo:ME - Muy Bajo, B - Bajo, M - Moderado, A - Alto, MA - Muy Alto.

Sistemas de información

Estructura

Dentro del sistema de información, se puede acceder a los recursos distribuidos en función de sus características y frecuencia de acceso.

```
/TFG
|_____ Memoria
|       |_____ Imágenes/
|       |_____ Memoria.doc
|
|_____ Afiche
|       |_____ Afiche.eps
|       |_____ Licencia.txt
|
|_____ DataSets
|       |_____ Train1.csv
|       |_____ Test1.csv
|
|_____ Planificación
|       |_____ Tareas.xlsx
|       |_____ Doc1.doc
|
|_____ Bibliografía
|       |_____ Título - Autor.pdf
|       |_____ ...
|
|_____ Otros archivos
        |_____ logo.svg
        |_____ Testing/
        |_____ EjParalelizacionPython.zip
```

Acceso

Código fuente: Se utilizará GitHub como gestor de versionado, un repositorio accesible de carácter público:

Interfaz: https://github.com/leroydeniz/GANN-FRONT
API: https://github.com/leroydeniz/GANN-API

Documentación: En cuanto a la documentación relacionada con los desarrollos y la memoria, será accesible a través del siguiente enlace a una carpeta en Google Drive: http://tiny.cc/cmlvtz.

Aplicación: la aplicación estará accesible a través de la web en el siguiente enlace:
https://gann.es

Formatos

Los DataSets con los juegos de datos de entrenamiento y testing para el algoritmo se almacenarán en csv.
Los modelos de definición de redes neuronales usarán el estándar de Machine Learning, onnx.
La documentación bibliográfica se almacenará en formato pdf.
Los archivos editables generados a raíz del desarrollo del trabajo serán en doc.
La memoria del trabajo también será en doc, editable desde Google Drive.
La planificación se desarrollará en Google Drive y se almacenará en formato xls.

Capítulo II

Definiciones tecnológicas

Definiciones tecnológicas

"Algunas veces hay que decidirse entre una cosa a la que se está acostumbrado y otra que nos gustaría conocer."

PAULO COELHO

En este apartado se identifican definiciones tecnológicas, permitiendo establecer un marco sobre el cual desarrollar la aplicación y su documentación. La necesidad de establecer criterios tecnológicos y procedimentales desde el inicio, permite no solamente una mejor estimación del tiempo necesario, sino que además contribuye en la definición del alcance, puesto que son decisiones definitorias y previamente acordadas en función de su potencial.

Formatos de ficheros

- Transporte de datos: JSON
- Conjunto de datos: CSV
- Modelo de Red Neuronal: ONNX
- GUI: HTML5, JS5 y CSS3

Software de servidor

- Servidor web: Nginx v1.14.2
- Servidor interfaz HTTP con Python: Gunicorn v20.1.0
- Lenguaje de API: Python v3.7.3
- Certificado SSL: Let's Encrypt v0.31.0

Aplicaciones

- Diseño: Photoshop e Illustrator v2021
- IDE: Visual Studio Code v1.58.2
- Transferencia de archivos por FTP: FileZilla v3.55.0
- Diseño UML: Creately

Frameworks

- Python framework: Flask v2.0.1
- GUI framework: React v17.0.2
- Entrenamiento RNA: PyTorch v1.9.0
- Manejo de volúmenes de datos: Pandas v1.3.0
- Herramientas de algoritmos matemáticos: Scipy v1.7.0
- Manejo de vectores y matrices: Numpy v1.21.0
- Construcción de Algoritmos Genéticos: deap v1.3.1
- Aprendizaje automático: Scikit-learn v0.24.2
- Creación de modelos de RNA: Onnx v1.9.0

Hardware mínimo

- Sistema operativo: Debian 10
- Memoria RAM: 8GB
- Procesador: 4 núcleos
- Espacio de disco: 100GB

Capítulo III

Marco teórico y antecedentes

Marco teórico y antecedentes

"En vez de intentar producir un programa que simule la mente adulta, ¿por qué no tratar de producir uno que simule la mente del niño? Si ésta se sometiera entonces a un curso educativo adecuado se obtendría el cerebro de adulto."

ALAN TURING

Sobre las Redes Neuronales Artificiales (RNA)

En el estudio de Romero (s. f.), se define una Red Neuronal Artificial (RNA) como un grafo dirigido, cuyos nodos se llaman elementos de proceso (EP), que pueden tener cualquier número de conexiones y memoria local, y poseen una función de transferencia que -en función de las señales de entrada y la información de su memoria local- producen una señal de salida o altera su propia información local. Estos enlaces se llaman conexiones y funcionan como caminos unidireccionales instantáneos.

Por su parte, las RNA constituyen para Rivas Santos (2003), un paradigma de la computación útil para la resolución de problemas complejos: reconocimiento del habla y de imágenes, planificación de movimientos, entre otros muchos. Constituye un modelo específico dentro de los modelos estadísticos. Otra definición con un carácter más matemático-informático es planteada por Hilera González (2021), que afirma que las redes RNA son sistemas computacionales compuestos con un gran número de neuronas artificiales, que procesan información dinámicamente como respuesta a entradas externas, devolviendo un resultado.

Intentando unificar y compactar las tres definiciones anteriores, podemos afirmar que una Red Neuronal Artificial es una estructura que permite aprender de la información que se le proporciona, generando una respuesta acorde al contexto que busca resolver.

Para Besogain (2021), las características principales de las RNA son el aprendizaje, capaz de adquirir conocimiento a través de la experiencia, entendiendo esta última como la información que se le suministra a la RNA a la hora de entrenar el modelo; la capacidad de generalización para permitir aproximar la respuesta a pesar de cierta inconsistencia en los datos, y la abstracción que dota a la RNA de la capacidad de encontrar patrones que faciliten la toma de decisiones.

En cuanto a su paralelismo biológico, las RNA son modelos que han surgido con la idea de intentar formalizar matemáticamente la estructura del cerebro, de tal forma que imitan la estructura hardware del sistema nervioso, basándose en el aprendizaje a través de la experiencia para lograr extraer conocimiento a partir de ella (Lopez y Fernandez, 2008). Para Izaurieta y Saavedra (2000), el cerebro humano corresponde al de un sistema complejo, no-lineal y paralelo, es decir, que es capaz de ejecutar un gran número de operaciones simultáneamente a diferencia de un ordenador clásico, que ejecuta de forma secuencial una serie de instrucciones pero una a la vez.

La estructura de una RNA consiste en un conjunto de neuronas artificiales cuya conexión pasa de etapa en etapa hasta generar una salida; es decir, el foco no está únicamente en lo que procesa cada una de las neuronas, sino que también en la forma en la que estas se conectan. Estas etapas se denominan capas, cada neurona pertenece a una y solo una de estas y además está vinculada únicamente con las neuronas de la capa anterior y con las de su siguiente, recibiendo información de las primeras y enviando su cómputo a las segundas.

Cada neurona artificial o elemento de proceso recibe varias entradas que combina, pondera y a

través de su función de transferencia, envía el dato a través de su salida que pasará a la siguiente capa.

Las RNA imitan el funcionamiento de las neuronas biológicas abstrayendo las estructuras neurobiológicas y se caracterizan por ser sistemas desordenados y poseedoras de unidades de información: "Formadas a través de una interconexión de redes, de forma paralela, estas redes poseen una organización jerárquica que permiten interactuar con el mundo, además se conoce como un sistema de computación constituida por un gran número de elementos simples de procesamiento muy interconectados, que procesan información en respuesta para algún estímulo externo" (Rivas y Mazón, 2018, p. 12).

Tal y como se conoce de la realidad biológica, cada neurona está compuesta por un Soma o cuerpo celular donde se realiza el procesamiento de la información; una gran cantidad de dendritas cuya finalidad reside en recibir la información proveniente del exterior; un axón por el cual se envía la información a enviar a las demás neuronas y donde finaliza una serie de botones sinápticos que son los puertos de envío de información al mundo. A grandes rasgos, podemos identificar estas partes como aquellas que nos interesan para el modelo que se quiere construir en una RNA.

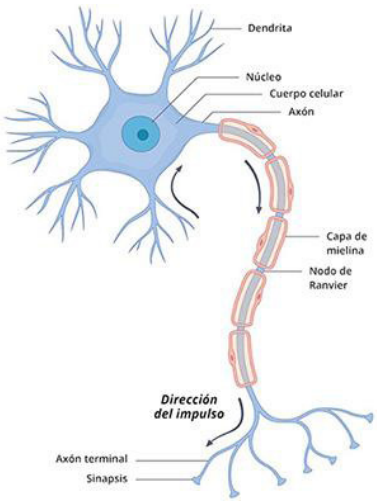


Figura 2: imagen de una neurona animal.[1]

Como dato cuantitativo, basándonos en el estudio de Izaurieta y Saavedra (2000), se estima que en un cerebro la cantidad de neuronas está en el entorno de 10^{10}, unas diez mil millones y cada una tiene un tiempo de procesamiento de unos pocos milisegundos (10^3), lo cual a través de una interconexión masiva entre ellas, parece compensar excepcionalmente el tiempo con el que una RNA que trabaja en un soporte de silicio, demora aproximadamente unos pocos nanosegundos. En la estructura comunicacional entre las neuronas, el cruce de información se da entre el botón sináptico y la dendrita, es decir, entre el final del axón, por donde corre el pulso de energía de la célula con la información procesada, hacia los puertos de recepción de información de otras neuronas.

Aproximándonos entonces hacia un modelo más matemático de este paralelismo, se podría abstraer esta información en el siguiente esquema, donde el elemento identificado como y_i es aquella neurona que interesa estudiar y que pertenece a una capa α. Además, se representan con x_i a las neuronas artificiales que pertenecen a la capa inmediatamente anterior a α y que le estarán enviando información procesada a la neurona en cuestión.

1 Fuente: https://espanol.nichd.nih.gov/salud/temas/neuro/informacion/partes

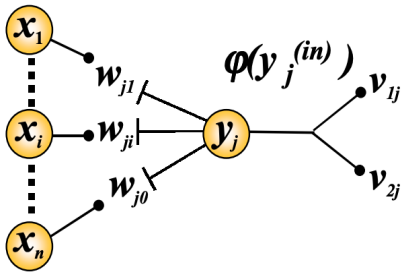


Figura 3: Perceptrón.[2]

Los valores enviados a la neurona y_i se representan como w_{ji} y representan los pesos sinápticos en cada una de las dendritas de la neurona receptora, siento el primer subíndice esta última y el segundo subíndice la emisora del dato. Cada w_{ji} multiplica su entrada x_i y pondera cada una, sumando todas estas. La siguiente ecuación muestra la forma de cálculo que implementa una red.

$$y_j^{(in)} = \sum_i^n w_{ij} \cdot x_i$$

Una vez se ha procesado la información de entrada de todas las neuronas de la capa anterior, la neurona debe aplicar su función de activación que puede ser, por ejemplo, sigmoidal, tangente hiperbólica o ReLU, aunque en este punto se seguirá generalizando y llamaremos a cualquiera de estas *f*. Se tiene entonces que la señal de salida de la neurona se calcula a través de la siguiente función.

$$y_j = f\left(\sum_i^n w_{ij} \cdot x_i\right) = f\left(y_j^{(in)}\right)$$

Las RNA requieren valores numéricos para sus neuronas y sus enlaces, y puede ser representado en un modelo conexionista considerando que es un conjunto de elementos de cómputo densamente interconectados (Rivas Santos, 2003). "El modelo se puede definir como un grafo ponderado, *G*, determinado por un conjunto de *m* nodos o unidades de procesamiento, *U*, y una topología, Λ , dada por el conjunto de aristas o conexiones entre las unidades".

$$G = \{ U, \Lambda \}$$

Cuando se manipulan grandes volúmenes de neuronas artificiales, es claro su organización en capas, que pueden distinguirse entre capas visibles y ocultas, siendo las visibles aquellas que interaccionan con el medio, correspondiéndose así U_1 y U_n a las capas o niveles de entrada y salida de la red respectivamente. Cada capa es un vector de neuronas. Todas las capas entre U_1 y U_n son ocultas. En la siguiente figura se muestra un ejemplo de RNA con tres capas ocultas de nueve neuronas cada una, una capa de entrada de ocho y una capa de salida de 4.

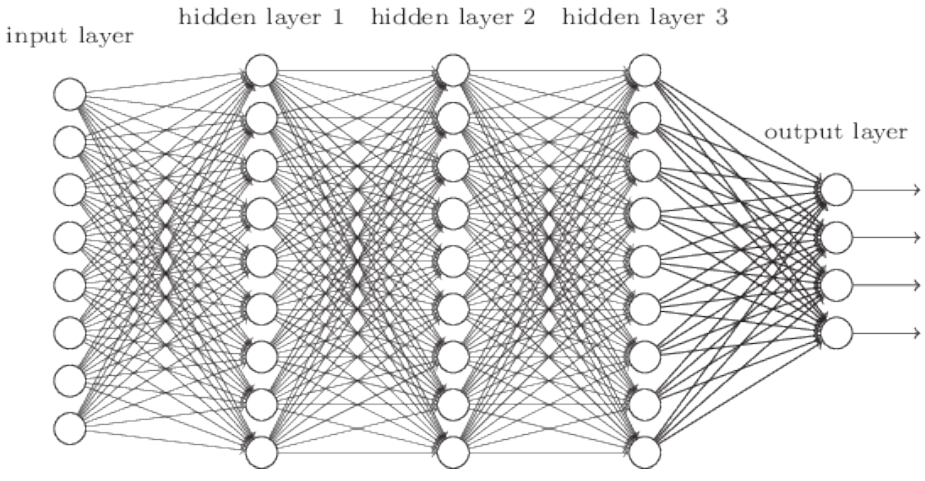


Figura 4: Ejemplo de red neuronal profunda.[3]

2. Fuente: https://disi.unal.edu.co/~lctorress/RedNeu/LiRna003.pdf
3. Fuente: http://www.cs.us.es/~fsancho/?e=165

En otras palabras, la arquitectura de una RNA se basa en una capa de entrada (input layer), a través de la cual se recibe el estímulo del exterior. Cada una de esas neuronas artificiales se está vinculando con todas las neuronas de la siguiente capa, lo que dará lugar al proceso de cálculo en cada una de ellas y el pesado para enviar ese dato a la siguiente capa. Una RNA puede tener tantas capas ocultas (hidden layers) como se quieran y tendrá también una única capa de salida (output layer), que dará el resultado que se está buscando a través de este modelo. Las redes neuronales tienen entradas que provienen del exterior, aunque su salida tiene información que se envía desde la propia RNA, a partir de su propio aprendizaje.

Ahora bien, entre la capa de entrada y la capa de salida se pueden incorporar capas ocultas, es decir, conjuntos de neuronas que tendrán un comportamiento similar y cuya finalidad es solventar las limitaciones del modelo simple, generando una red neuronal que se denomina perceptrón multicapa (Larrañaga et al., s.f.).

El Deep Learning es una particularidad dentro de las redes neuronales, donde se tienen una serie de capas ocultas, permitiendo el procesamiento de los datos de manera jerárquica, mejorando así la significatividad de las representaciones, mejorando capa a capa.

Luego se tiene las funciones de activación que son las formas de transmitir la información de una neurona a la siguiente capa a través de su conexión de salida. Estas pueden modificar el valor que se transmite según qué función se haya definido, aunque se utilizan para efectivizar la no-linealidad favoreciendo a la red en su capacidad de resolver problemas de complejidad aún más alta (Molino, Cardoso, Ruíz y Sánchez, 2014, pág. 1). Las funciones de activación más utilizadas están aglomeradas en cinco familias que se identifican a continuación:

1. Función identidad (identity). Sea cual sea el valor que deba propagar la neurona artificial, esta propaga el mismo valor sin alterarlo. La gráfica que representa esta curva es la siguiente.

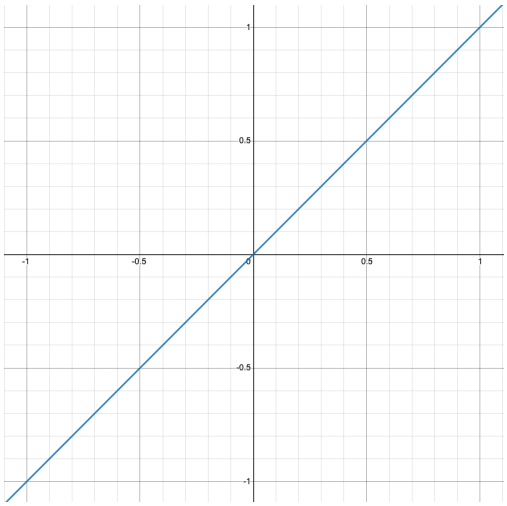


Figura 5: Función identidad $f(x) = x$

2. Función escalón (threshold). Implica una clasificación estricta donde cualquier valor ponderado negativo es propagado como un 0, mientras que todo valor positivo incluyendo el 0 es propagado como 1. Es una función sin valores intermedios, de tipo binaria y es considerada la función más rígida de las familias. La función se acota entre 0 y 1 y la gráfica que representa esta curva es la que se muestra a continuación.

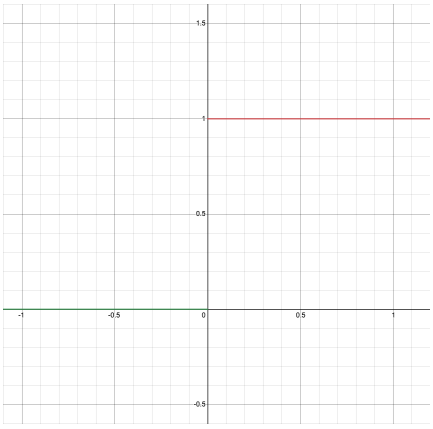


Figura 6: Función escalón $f(x) = 0 \text{ if } (x < 0)$
$1 \text{ if } (x \geq 1)$

3. Función sigmoide (sigm). Tiene una similitud con la función escalón, pero en esta sí existen valores intermedios entre los distintos valores que puede tomar, siendo menos estricta que la primera. La función es sensiblemente más suave que la anterior. Cuanto más negativo es el valor, más se aproxima a 0 y cuanto más positivo más cerca está del 1. La función se acota entre 0 y 1 y la gráfica que representa esta curva es:

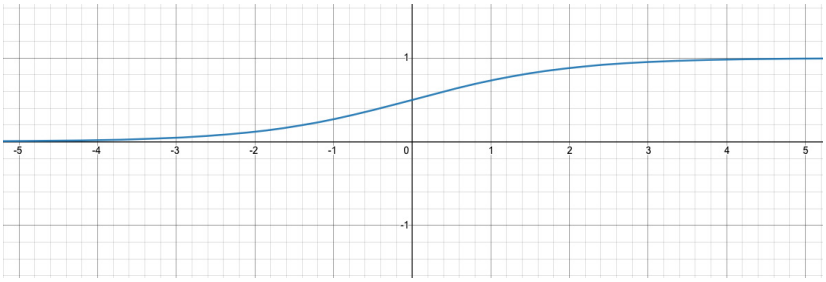


Figura 7: Función sigmoide $f(x) = 1 \,/\, (1 + e^{-x})$

4. Función rectificadora (ReLU). Esta función toma el valor 0 para cualquier valor negativo de entrada, mientras que devuelve la identidad cuando el valor es positivo. Esta función, a diferencia de las dos anteriores, no está acotada. La función que representa esta curva es la que se muestra a continuación.

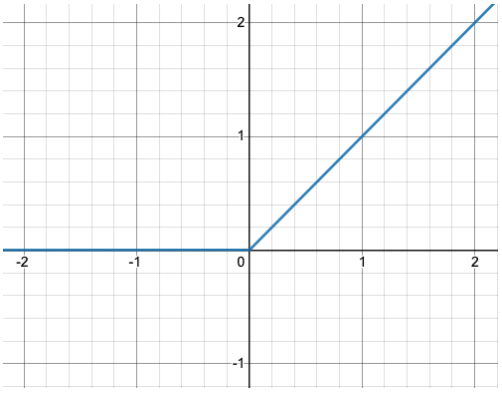


Figura 8: Función ReLU $f(x) = max(0, x)$

5. Función tangente hiperbólica (tanh). Esta función es muy similar a la sigmoidal aunque está acotada entre -1 y 1, pero sigue la misma lógica que su similar, para los valores más negativos, devolverá un valor muy próximo a -1 y para aquellos valores más positivos, devolverá un valor muy próximo al 1. La gráfica que representa a la función es la que se muestra a continuación.

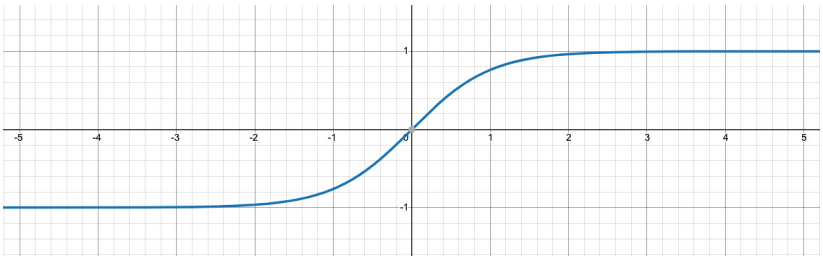


Figura 9: Función tanh $f(x) = (1 - e^{-2x}) / (1 + e^{-2x})$

Cada una de las capas puede tener su propia función de activación en función de aquella que se ajuste más al problema. Si no hay restricciones en el rango utilizado, puede utilizarse la función identidad para enviar a la siguiente capa el valor que ha recibido; por el contrario, si interesara devolver un valor binario, convendría utilizar la función escalón que, como se ha definido previamente, devuelve únicamente estos dos valores en función de su entrada; en caso de utilizar derivadas habría que utilizar una función de activación donde la derivada esté siempre definida, aunque estas últimas son más costosas (García, 2020).

Para dar una idea breve de las ventajas que ofrece una RNA y en base a la estructura que se ha ido definiendo anteriormente, podemos identificar algunas de estas como (a) autoorganización que le permite a la propia RNA representar la información de entrada en su etapa de entrenamiento; (b) aprendizaje adaptativo, lo que le permite ejecutar procesos en función de su experiencia; (c) tolerancia a fallos, la alta densidad de conexiones entre las neuronas de las capas permite mantener la integridad de la red incluso tras un gran daño en esta; (d) operación en tiempo real, permite realizar cómputos en paralelo utilizando hardware especializado, (Matich, 2001).

Entonces, las RNA son herramientas muy poderosas para analizar las señales y la modelado de sistemas, ya que son capaces de resolver funciones altamente no lineales en un tiempo corto porque aprenden de los datos que son difíciles de expresar matemáticamente.

Sobre los Algoritmos Genéticos (AG)

"Este principio, por el cual toda ligera variación, si es útil, se conserva, lo he denominado yo con el término de selección natural."

CHARLES DARWIN
El origen de las especies, 1859.

Para Rivero Cides (2019), quien se basa en la definición de F. Glover, las metaheurísticas son procesos iterativos que conducen a la heurística en la exploración del espacio de búsqueda. Este tipo de algoritmos no garantizan el encontrar el óptimo global de este espacio, pero sí aproximan muy bien a una solución lo suficientemente buena aunque esta pudiera ser local.

La metaheurística que se conoce como Algoritmos Genéticos (AG), son métodos utilizados para búsqueda y optimización que, simulando la evolución natural de los seres vivos, parten de poblaciones para evolucionar, reproduciéndose y mutando bajo ciertas condiciones, siguiendo la teoría de Darwin. Tal y como sugiere Calegari (2002), es una herramienta de mucha utilidad de cara a atacar problemas de tipo NP-completos, puesto que ofrecen soluciones flexibles a problemas complejos.

También son definidos por Kuri (2000) cuando los propone como un método para la búsqueda de soluciones óptimas, que imitan la evolución natural en varios tipos de problemas de optimización combinatoria, de funciones reales y para inteligencia artificial. Es a partir de esta última característica que se usarán en este trabajo para hacer evolucionar una familia de redes neuronales. Estos algoritmos han sido desarrollados por John Holland y algunos de sus alumnos de la Universidad de Michigan en los años 70.

Otra definición más o menos similar la propone Hidalgo Sánchez y Turrado Martínez (s. f.), cuando afirma que los AG "son técnicas de resolución de problemas de optimización, que se basan en emular la HERENCIA GENÉTICA y el PRINCIPIO DE SUPERVIVENCIA".

En una manada de ciertos animales hay algunos más rápidos que otros, hay algunos con más tendencia a enfermarse, otros más débiles, y una serie de características que identifican a cada individuo; digamos a este punto que cada uno de estos individuos será parte del algoritmo genético. Todas estas diferencias pueden ser consideradas relativas ya que depende directamente de los individuos que se estén comparando a la vez. Sería válido decir "este individuo es más veloz que el resto de la manada" o bien "este es el más sano de todos", aplicando una comparativa sobre una muestra o el Universo que se está estudiando (Kuri, 2000).

En una población de depredadores que cazan a otra población de presas, es importante destacar el criterio con el que se van a permitir soluciones buenas o no. En un ejemplo más concreto y siguiendo la línea de razonamiento de Kuri, una población de gacelas que es cazada por un cheetah, habría que aclarar que el criterio de supervivencia está definido por aquellas gacelas cuya velocidad supere al del cheetah, caso contrario la población entera desaparecería. En este ejemplo y al igual que en los algoritmos genéticos, este tipo de reglas condicionan la selección de los individuos a la hora de establecer su selección: "gana el más fuerte y el mejor adaptado".

En un claro paralelismo con la Naturaleza, donde los individuos compiten entre sí por los recursos disponibles como comida, agua, entre otros muchos, según el Darwinismo sobreviven aquellos que mejor se adaptan al entorno, los más fuertes y mejor preparados. Por el contrario, aquellos individuos que no estén tan preparados o no evolucionen de la manera adecuada, tenderán a desaparecer o al menos a tener menos descendientes. Naturalmente esto hace que sean los mejor adaptados los que definan la continuación de la especie y la lucha por los recursos sea cada vez más fuerte. Combinando las mejores características de cada padre, se irán propagando de generación en generación mejorando significativamente la adaptación (Hidalgo Sánchez & Turrado Martínez, s. f.).

Para Kuri (2000) los AG son una descripción de la naturaleza, que a través de pequeñas evoluciones va generando individuos cada vez mejores y más adaptados al entorno con sus mil características. Toma poblaciones cuyos hijos mejoran respecto a sus progenitores aunque quizá de manera muy paulatina pero constante.

Estos algoritmos son buenos para resolver problemas cuyas soluciones especializadas no existen aún, aunque también pueden utilizarse como herramientas de mejora para aquellas técnicas que ya de por sí son buenas, optimizándolas aún más.

Como bien afirma Estévez Valencia (1997) estos algoritmos son eficientes a la hora de encontrar individuos que minimicen o maximicen una función de adaptación de adaptación (fitness).

La estructura de un Algoritmo Genético Simple puede definirse en una serie de pasos, que permiten entender la situación de partida de la especie y el entorno, y bajo qué circunstancias irán apareciendo las nuevas generaciones. Es por ello que se puede definir al AG con los siguientes parámetros:

```
t=0         instante de tiempo inicial
P(t)        definición de la población inicial
Eval( P(t) )      evaluar la mejora de población
Iterar hasta condición de finalización:
    t = t + 1
    Selección P(t-1) → P(t)
    Cruce P(t)
    Mutación P(t)
    Evaluar P(t)
```

En este tipo de algoritmos a diferencia de otros de optimización, no trabaja sobre individuos particulares, sino sobre familias de individuos que luego seleccionará, cruzará y mutará.

Los AG tienen tres tipos de operadores o etapas de evolución, que permiten elegir, reproducirse y mutar a los padres en busca de descendencia evolucionada. A continuación se identificarán los tres operadores importantes en la evolución y brinda brevemente una explicación sobre cada uno de ellos.

Selección

En cada etapa del algoritmo, la población debe ser ordenada en función de un determinado criterio a través del cual serán elegidos los mejores de esta población para continuar con la reproducción. Se puede ejemplarizar esta condición suponiendo que en una etapa, todos los individuos seleccionables deberán tener una calificación mayor o igual a □, luego serán estos mismos quieren se reproduzcan dando lugar a nuevos individuos evolucionados cuya calificación mínima será al menos □. Podría considerarse que se ha mejorado la población.

El problema de la selección únicamente de los individuos cuyo criterio sea el mejor, es que la población convergerá hacia él, pudiendo no ser necesariamente el óptimo global, ocasionando así el efecto convergencia prematura. Una estrategia para evitar tener una población que se vicie es elegir en total los mejores n mejores elementos de la vieja generación para pasar intactos a la siguiente, llamándose a este criterio Elitismo. Puede incluso establecerse que pasen los n mejores individuos de las últimas k generaciones.

Otra forma de elección es la Rueda de ruleta, que permite al algoritmo seleccionar al azar individuos que pasarán a la siguiente generación, de forma que permite a este dar saltos por el dominio del problema, pudiendo alcanzar así a un óptimo global con mayor certeza.

Cruce o reproducción

Una vez se alcanza la etapa de reproducción significa que ya han sido seleccionados los individuos que se cruzarán, por lo que es necesario ahora definir cómo se forman los hijos con

la información de sus padres. Kuri (2000) utiliza el ejemplo de las gacelas para ilustrar mejor esta etapa: “es posible que una gacela herede la velocidad de su abuelo paterno y la fuerza de su abuela paterna, la salud de su abuelo materno y la agudeza visual de su abuela materna. Si estas características le confirieron a sus ancestros una alta aptitud de supervivencia, entonces este individuo será, con alta probabilidad, un individuo exitoso en su manada”.

Existen varios criterios para cruzar la información de los padres de cara a ser heredados por los hijos, uno de ellos es Crossover 1 punto, que se basa en partir a sus padres por un punto y formar a cada hijo con una parte de cada uno de sus padres, de tal forma que cada uno de ellos esté compuesto por una parte de los dos y lo importante es definir cuál es ese punto en el que se van a dividir los padres.

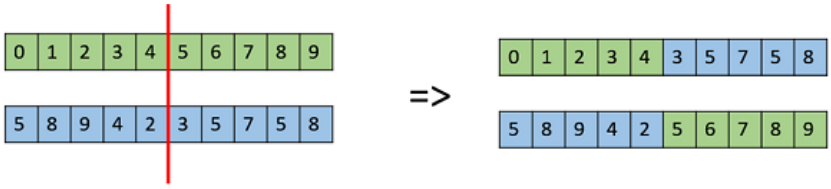


Figura 10: Crossover 1 punto[4]

Otra alternativa para el operador de cruce es la técnica Crossover 2 puntos que, de la misma forma que la primera, divide a los padres en tres partes, intercambiando el material genético de estos para que ambos hijos tengan material de los dos progenitores.

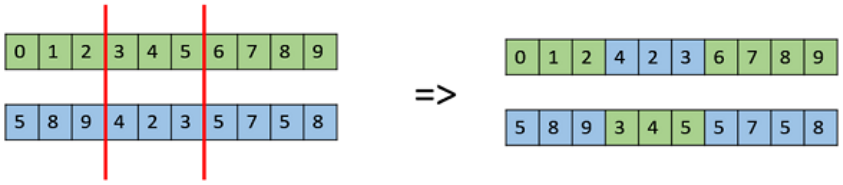


Figura 11: Crossover 2 punto[5]

Mutación

Una vez se tienen los hijos con la información genética heredada, es momento de generarles una mutación de cara al proceso de evolución. Estos hijos son, a pesar del intercambio genético, copias de sus padres y por lo tanto mantendrán las mismas cualidades. El proceso de mutación se trata de inducir un cambio en los hijos que permita mejorar la adaptación de estos al medio.

Para aquellos individuos que estén formados por genes binarios, puede aplicarse la técnica de Switch que requiere cambiar el valor booleano de uno de sus genes por su opuesto. La técnica que permite insertar un nuevo elemento en los hijos se denomina Insert y la que permite intercambiar dos elementos de cada hijo se llama Swap. Estos cambios buscan mejorar a los hijos con respecto a sus padres.

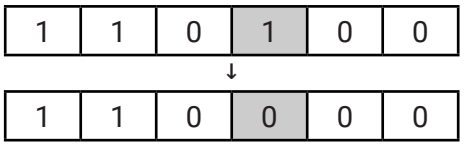


Figura 12: Operador Switch

4. Fuente: https://bit.ly/2TdfXIV
5. Fuente: https://bit.ly/2TdfXIV

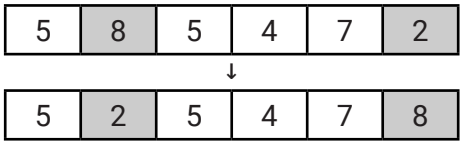


Figura 13: Operador Swap

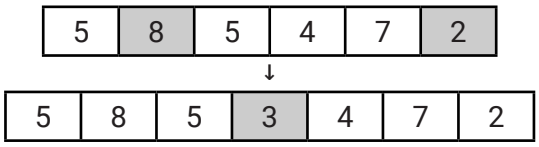


Figura 14: Operador Insert

El AG Simple sigue una estructura básica que se presenta a continuación a través de un pseudocódigo.[6]

```
BEGIN /* Algoritmo Genetico Simple */
  Generar una poblacion inicial.
  Computar la funcion de evaluación de cada individuo.
  WHILE NOT Terminado DO
  BEGIN /* Producir nueva generación */
    FOR Tama no poblacion/2 DO
      BEGIN /*Ciclo Reproductivo */
        Seleccionar dos individuos de la anterior generación,
        para el cruce (probabilidad de selección proporcional
        a la función de evaluación del individuo).
        Cruzar con cierta probabilidad los dos
        individuos obteniendo dos descendientes.
        Mutar los dos descendientes con cierta probabilidad.
        Computar la función de evaluación de los dos
        descendientes mutados.
        Insertar los dos descendientes mutados en
        la nueva generación.
      END
      IF la poblacion ha convergido THEN
        Terminado := TRUE
    END
END
```


6. Fuente: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

Sobre el diseño de RNAs evolucionadas por AGs

"No es el más fuerte de las especies el que sobrevive, tampoco es el más inteligente el que sobrevive. Es aquel que es más adaptable al cambio."

CHARLES DARWIN
El origen de las especies, 1859.

Tal como se ha mostrado en los apartados anteriores, las RNAs son una herramienta poderosa para la clasificación supervisada, más ahora donde el poder de cómputo y la memoria disponible ya ha dejado de ser un problema. Por otro lado, los AGs son una metaheurística que permitirá evolucionar la RNA en busca de un óptimo global, permitiendo así mejorar considerablemente la bondad de estas.

Para Castillo Valdivieso et al. (2000), el mayor problema a la hora de utilizar este tipo de técnicas, está en elegir correctamente los parámetros que permitan alcanzar un modelo que logre representar los datos con los que se entrena. Es decir, para el modelo de perceptrón multicapa (PMC) que aquí se desarrolla, lo complejo es encontrar la cantidad correcta de capas, el tamaño adecuado para cada una de estas y los pesos apropiados. Tal es así que a pesar de haberse ideado una gran cantidad de algoritmos que permiten optimizar estos parámetros, se suele caer en óptimos locales.

Para Rivas Santos (2003), la aplicación de RNA que busca resolver un problema requiere establecer algunos parámetros, que influyen significativamente en la velocidad de aprendizaje y en la capacidad de aproximación de la propia red. De toda la familia de RNA que se puede obtener, solo algunas de ellas serán capaces de resolver la clasificación de manera inmejorable, por eso la aplicación de algoritmos evolutivos hace de este un problema de optimización en pro de encontrarlas.

Estos heurísticos suelen no ser rápidos en términos de tiempo, ya que realizan una exploración del espacio de búsqueda, pudiendo en ocasiones aparcar en óptimos locales. Para evitarse este problema, es importante elegir bien los operadores que el AG utilizará de forma que no aparque en estos y sea capaz de encontrar el óptimo global más allá, si es que lo hay. El enfoque que recibe este tipo de algoritmos, que se basan en la combinación de dos, se llama híbrido y con una buena configuración permite mejorar sustancialmente las métricas.

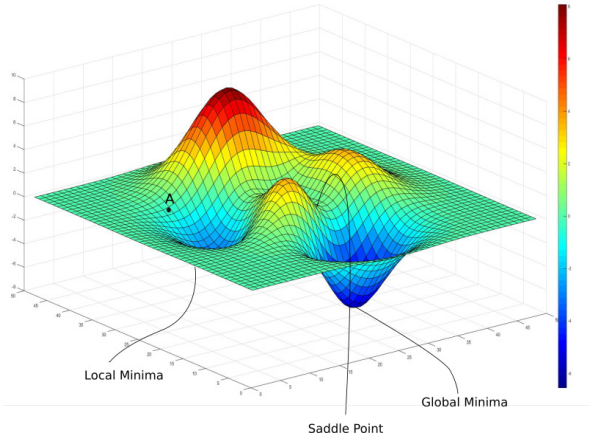


Figura 15: Mínimo local y global de la función.

Para establecer los pesos correspondientes a las conexiones entre las neuronas, se plantea como un problema de minimización utilizando una función de error, en este trabajo se utilizará

específicamente el algoritmo de back-propagation, que se basa en descenso de gradientes. Esos algoritmos, como backpropagation, tienen el inconveniente de apartarse en mínimos locales cuando la función no es diferenciable. Cuando se utiliza esta combinación se soluciona el problema de estancamiento en extremos locales, ya que la función fitness incluye factores relativos a los errores. De cara al éxito del proceso de aprendizaje se basa en una correcta elección del esquema de representación de los operadores del algoritmo evolutivo y sus soluciones, creando una nueva red a partir de estos datos en función de su valor fitness y evolucionando hasta que se alcance la condición de parada o el máximo de iteraciones.

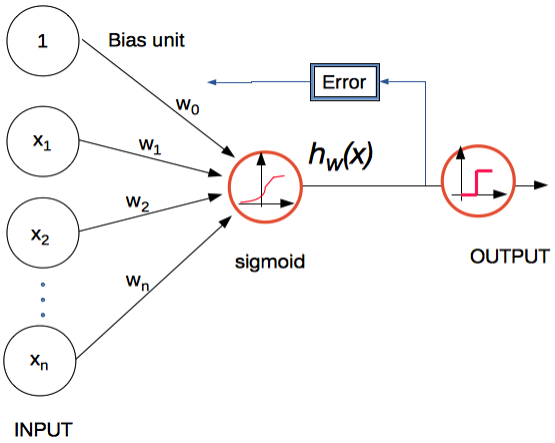


Figura 16: Síntesis del Algoritmo de Back-propagation

El problema de asignación de pesos se conoce como problema de permutación, donde la configuración de los operadores de reproducción y mutación pueden tener una eficacia muy baja. Tal es así que varias redes, y sobre todo aquellas de codificación binaria, pueden ser distintas pero evaluar a través de la red de forma equivalente.

La eficiencia del trabajo de la RNA es muy dependiente de su topología, si no tuviera el suficiente número de capas o nodos, podría no ser capaz de aprender correctamente los patrones debido a su limitación de capacidades. Sin embargo, el exceso de capas y nodos no lineales puede sobre-entrenarse y caer en overfitting, lo que permitiría a la red entender a la perfección sus datos de entrada logrando una clasificación perfecta pero nada buena a la hora de reconocer los nuevos casos que nunca ha visto; en otras palabras, no es capaz de generalizar.

Según Miller et al. (1989), los algoritmos evolutivos como los genéticos, son candidatos perfectos para la búsqueda de soluciones óptimas globales por cinco principales razones:

a) El espacio de búsqueda es infinitamente grande, dado que no se debe establecer límite al número de posibles nodos y conexiones.

b) El espacio no es diferenciable con respecto a número nodos y conexiones, pues los cambios en tales valores son discretos y a su vez pueden provocar un efecto de discontinuidad en la salida ofrecida por la red.

c) Es un espacio complejo y en el que aparece ruido; dado que la codificación en un cromosoma de una determinada arquitectura impide relacionar directamente a está con su comportamiento, hace que surjan efectos de fuerte epistasia y que sea dependiente del método de evaluación utilizado.

d) El espacio de búsqueda es engañoso. Como ya se ha comentado, arquitecturas similares pueden dar lugar a resultados muy distintos con respecto al objetivo buscado.

e) El espacio de búsqueda es multimodal. Al contrario del punto anterior, arquitecturas muy diferentes entre sí pueden tener comportamientos muy parecidos.

Pacheco y Aragón (2001) también plantean una solución de entrenamiento de redes neuronales utilizando metaheurísticas que busca determinar los pesos de cada conexión entre neuronas, de cara a minimizar el Error cuadrático medio (E). También afirma que una de las limitaciones de los algoritmos como back-propagation es la convergencia a óptimos locales no globales y la dependencia directa de la solución inicial.

Calle Cárdenas (2014) concluye que una combinación de estos dos modelos es capaz de evolucionar RNA partiendo de topologías mínimas, complejizando la estructura a medida que se va necesitando, lo que lleva a que los pesos vayan modificándose conforme se incremente la estructura de las redes, entrenando las redes a la vez que se optimiza su estructura.

Capítulo IV

Dominio del problema

Dominio del problema

"Solo si nos detenemos a pensar en las pequeñas cosas llegaremos a comprender las grandes."

JOSÉ SARAMAGO
Ensayo sobre la ceguera, 1995

Esquema general de la aplicación

Para la implementación de la aplicación, se utilizará un diseño en tres capas como se define en la imagen a continuación, acoplando la lógica de negocio con la API de un lado de la infraestructura, y por otro lado la GUI que estará implementada en un hosting web compartido.

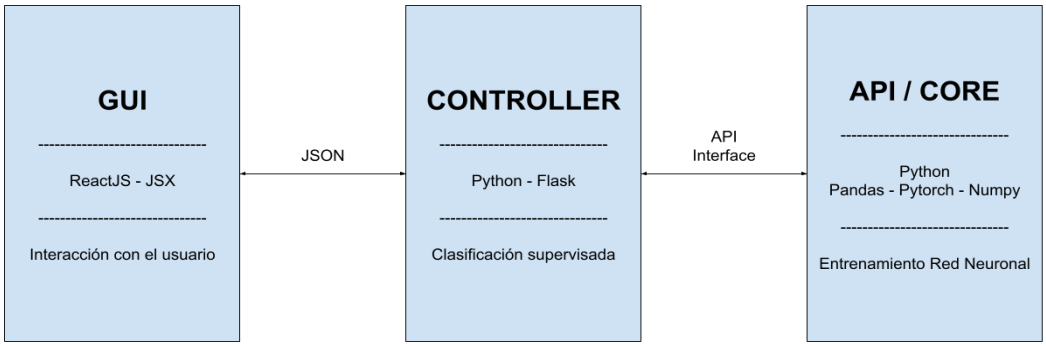


Figura 17: Estructura de la aplicación

Modelo de Componentes

Para construir la interfaz de vinculación con la API de entrenamiento, se define React como la tecnología a utilizar, siendo esta una librería de JavaScript que permite la implementación de sitios web en una única página con una fuerte integración de AJAX. En los apartados siguientes se detalla el por qué de la elección y su comparativa con otras tecnologías. En un modelo de tipo MVC, React es la Vista (V).

Se basa en un paradigma de orientación a componentes, por lo que en una única aplicación se navegará a través de Routers que le indicarán a la vista qué componente mostrar, y le permitirá vincularse a la capa de negocio según cuál sea el caso de uso que se está ejecutando. Es importante en este punto definir entonces el flujo de estos componentes y de qué manera estarán siendo invocados en función de sus componentes previos. Todos los componentes heredan de una superclase React.Component, esta clase no se muestra en el diagrama para facilitar la lectura y el entendimiento por parte del lector.

No hay un componente específico que sirve de bridge entre la vista y el controlador, sino que este vínculo es invocado a través del propio componente del caso de uso. Una vez la API responde, se mostrará dentro de esta misma vista la información relativa a los resultados, así como la posibilidad de exportar en determinado formato.

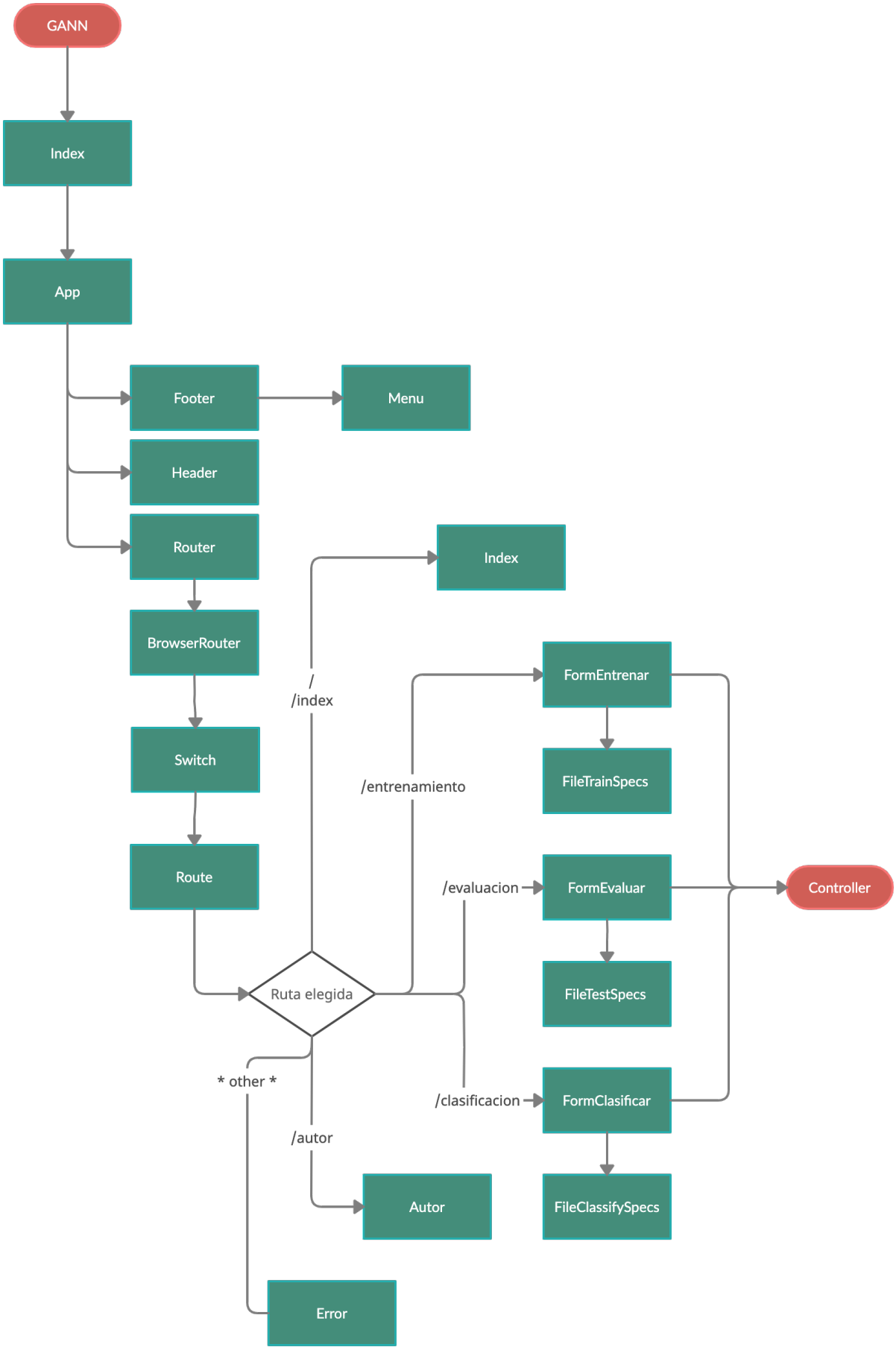


Figura 18: Modelo de componentes

Modelo de casos de uso

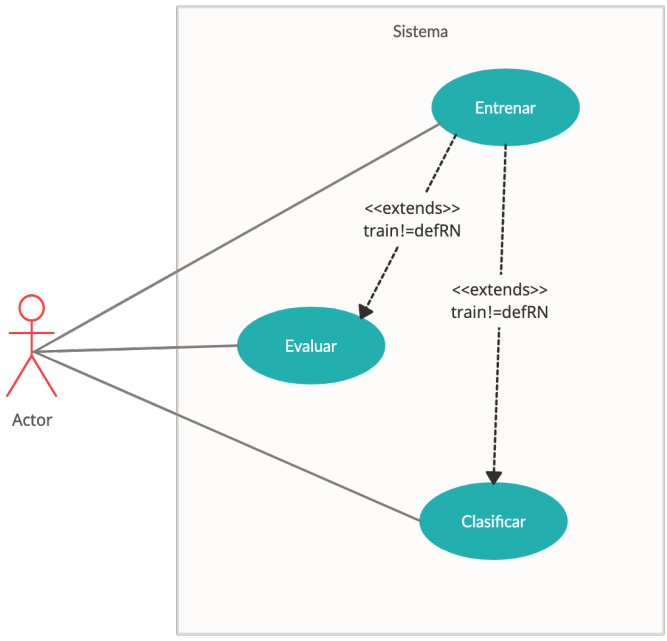


Figura 19: Modelo de casos de uso

Flujo de eventos

Caso de uso: Entrenar clasificador

Casos de uso vinculados: --
Actores: Usuario
Precondiciones: --
Descripción del flujo normal:
1. El usuario carga un conjunto de datos (dataset) de train a través del cliente web.
2. El dataset es enviado al Controller para su verificación a través de un webservice.
3. Una vez verificado, el Controller se comunica con la API, enviándole el dataset.
4. La API entrena la red neuronal con el DataSet recibido, devolviéndole al Controller la configuración de una red neuronal y otro documento con las métricas de evaluación.
5. El Controller devuelve esta configuración al cliente web y fuerza su descarga.
Descripción del flujo alternativo:
1.1. El usuario carga un DataSet para clasificar a través del cliente web.
2.1. El DataSet de clasificación es enviado al Controller para verificación a través de un webservice.
5.1. El Controller utiliza la configuración de red neuronal obtenida en pasos anteriores para clasificar el segundo DataSet, devolviéndole al cliente web un archivo CSV con el DataSet clasificado. (Caso de uso Clasificar dataset)
Postcondiciones: -
Resultado: Se devuelve al usuario la definición de una red neuronal entrenada.

Caso de uso: Evaluar clasificador con dataset de test

Casos de uso vinculados: -
Actores: Usuario
Precondiciones: -
Descripción del flujo:
1. El usuario carga la definición de una red neuronal y un DataSet clasificado.
2. El usuario elige el tipo de validación que quiere, ingresando la configuración necesaria (en caso de que no use los valores por defecto) y elige además el formato de archivo que espera (JSON/XML/CSV).
3. Tanto la configuración como el DataSet clasificado son enviados al Controller para su verificación estructural del DataSet y, una vez validados, son enviados a la API.
4. La API validará el clasificador con la configuración seleccionada, construyendo un archivo y devolviéndole al Controller este archivo.
5. El Controller le devolverá al cliente web el archivo cuyo contenido será impreso en pantalla y forzada su descarga en el ordenador cliente.
Postcondiciones: -
Resultado: Se devuelve al usuario un documento con métricas de evaluación.

Caso de uso: Clasificar dataset

Casos de uso vinculados: -
Actores: Usuario
Precondiciones: -
Descripción del flujo:
1. El usuario carga la configuración de una red neuronal y un DataSet sin clasificar.
2. Tanto la configuración como el DataSet son enviados al Controller para su verificación estructural y, una vez validados, son enviados a la API.
3. La API clasificará el DataSet con la configuración de la red neuronal recibida, construyendo un archivo CSV y devolviéndoselo al Controller.
4. El Controller le devolverá al cliente web el archivo cuyo contenido será impreso de manera compacta en pantalla y forzando su descarga en el ordenador cliente.
Postcondiciones: -
Resultado: Se devuelve al usuario un documento con el DataSet clasificado.

Diagramas de secuencia

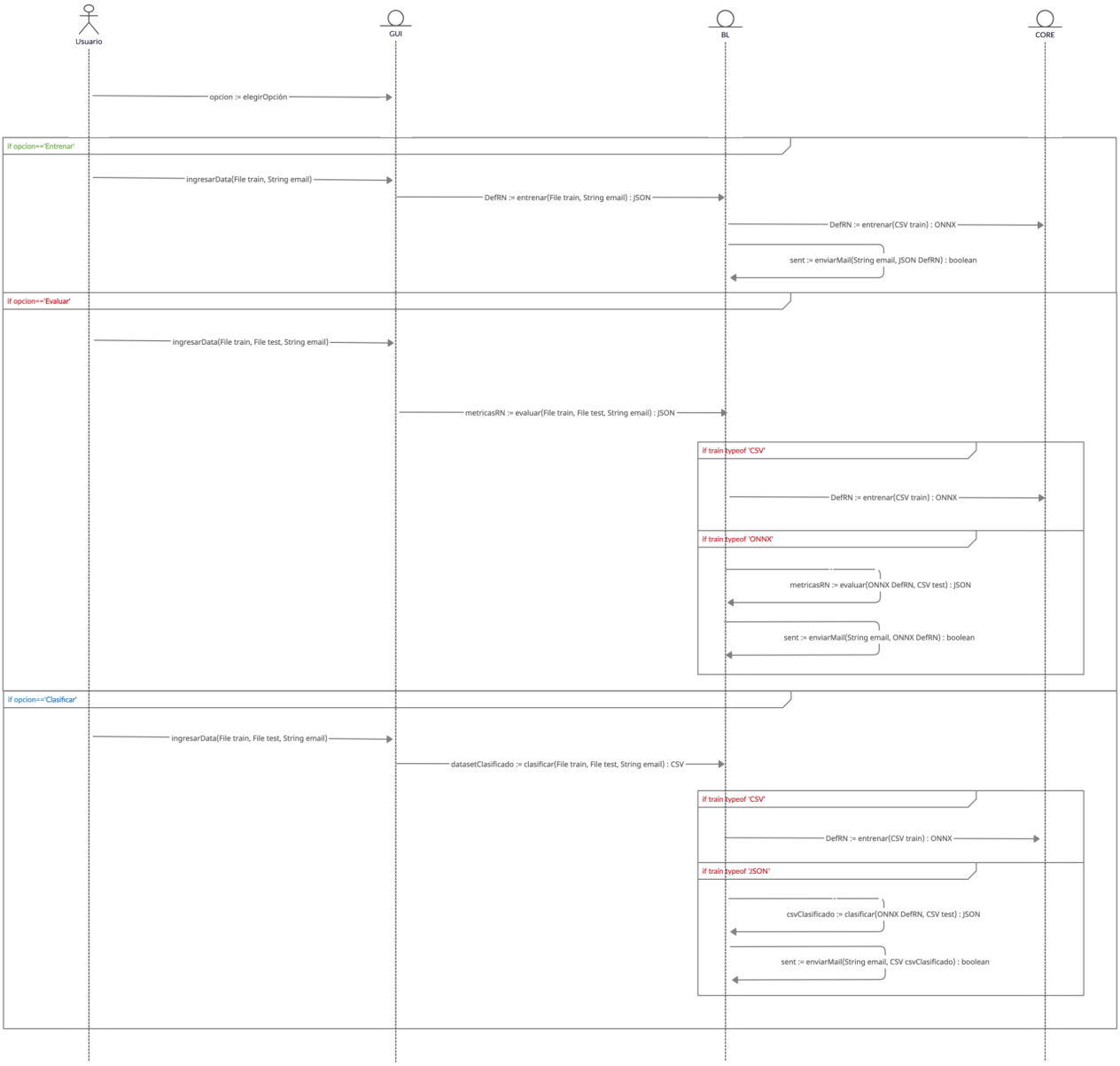


Figura 20: Diagrama de Secuencia de Sistema

Estructura de los datos de entrada

Los documentos que este software espera según el caso de uso pueden ser, o bien archivos CSV cuando de conjuntos de datos se trate, o bien archivos ONNX cuando se quiera enviar la definición de una red neuronal.

Ejemplo de conjunto de datos en CSV

```
"sepal.length","sepal.width","petal.length","petal.width",  "variety"
          5.1,          3.5,          1.4,          .2,  "Setosa"
          4.9,            3,          1.4,          .2,  "Setosa"
          4.7,          3.2,          1.3,          .2,  "Setosa"
          4.9,          2.4,          3.3,           1,  "Versicolor"
          6.6,          2.9,          4.6,         1.3,  "Versicolor"
          5.2,          2.7,          3.9,         1.4,  "Versicolor"
          6.9,          3.1,          5.4,         2.1,  "Virginica"
          6.7,          3.1,          5.6,         2.4,  "Virginica"
          6.9,          3.1,          5.1,         2.3,  "Virginica"
```

Definición de Red Neuronal en ONNX

A continuación se muestra un ejemplo gráfico de una red neuronal que se imprime partiendo del esquema que se importa desde el ONNX a través de la aplicación Netron.

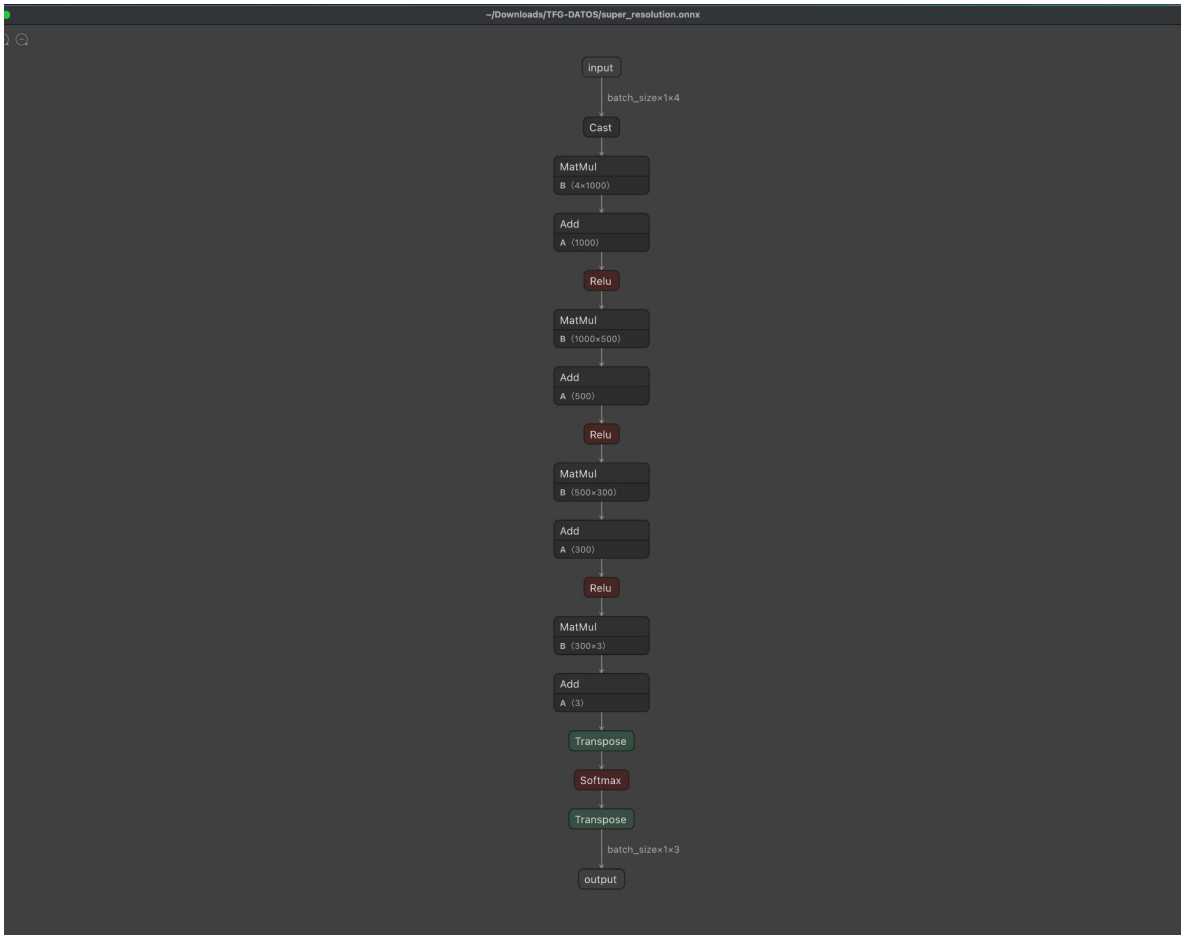


Figura 21: Esquema de la Red Neuronal de tres capas ocultas de 1000, 500 y 300 neuronas respectivamente

Capítulo V

Módulo de API

Módulo de API

"Y manos a labor; que en la tardanza dicen que suele estar el peligro."

MIGUEL DE CERVANTES
Don Quijote de La Mancha, 1605

Implementación del entorno

Para la implementación del servidor, se utilizó un Servidor Privado Virtual (VPS) accesible desde la url configurada en un vhost miescher.csic.edu.uy y bajo el alias https://api.gann.es. Las características del servidor sobre el que se implementa son las siguientes:

Sistema operativo:	Debian 10
IP:	164.73.68.59
RAM:	8GB
Procesador:	CPU(s)8 x Intel(R) Xeon(R) CPU E5504 @ 2.00GHz (2 Sockets)
Disco:	100GB HDD en LVM

Se configuran las medidas de seguridad necesarias para asegurar la integridad y la consistencia del servicio. El usuario de trabajo es gann. Es necesario la instalación a través del terminal, de los paquetes iptables para la configuración del firewall y fail2ban para evitar los ataques de fuerza bruta.

```
aptitude install iptables fail2ban
```

Es necesario agregar /sbin a la variable de entorno $PATH. Los únicos puertos accesibles son el 22 (ssh), 25 (smtp) 80 (http), 443 (https), 465 (smtp sobre ssl) y 5000 (flask), ejecutando el siguiente script de bash ubicado en /root.

```
#!/bin/sh

# DROP es política por defecto IPv4
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# DROP es política por defecto IPv6
ip6tables -P INPUT DROP
ip6tables -P FORWARD DROP
ip6tables -P OUTPUT DROP

# Permitir el tráfico en localhost
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Permitir todo lo relacionado o establecido
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

# Permitir paquetes ICMP
iptables -A INPUT -p icmp -j ACCEPT
iptables -A OUTPUT -p icmp -j ACCEPT

# Permitir conexión SSH a servidores internos y externos
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT

# Permitir peticiones de las aplicaciones web
iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 5000 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 5000 -j ACCEPT

# Permitir envío de mail
iptables -A OUTPUT -p tcp --dport 25 -j ACCEPT
iptables -A OUTPUT -p tcp --dport 465 -j ACCEPT
iptables -A INPUT -p tcp --dport 25 -j ACCEPT
iptables -A INPUT -p tcp --dport 465 -j ACCEPT

# Permitir resolución DNS
iptables -A OUTPUT -p tcp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT

# Permitir sincronización NTP
iptables -A OUTPUT -p tcp --dport 123 -j ACCEPT
iptables -A OUTPUT -p udp --dport 123 -j ACCEPT
```

Se crea además un script de bash para facilitar el borrado del firewall en caso de ser necesario. Su ubicación es la misma que el anterior.

```
#!/bin/sh
echo “Borrando reglas de firewall y permitiendo todo...”
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
```

Una vez ejecutado el script firewall.sh con permisos de root, es necesario persistir las reglas por cualquier reinicio del servidor, para eso se utiliza el paquete iptables-persistent que está disponible en los repositorios de Debian. Cualquier cambio en la política de firewall, basta con ejecutar los script de limpieza y carga de reglas y configurar una vez más iptables-persistent a través del siguiente comando.

```
dpkg-reconfigure iptables-persistent
```

Por seguridad se deshabilita también la conexión por IPv6 editando el archivo /etc/sysctl.conf, añadiendo las siguientes líneas de código al final y aplicando los cambios al finalizar con sysctl -p. Desde iptables ya fueron cerradas a través de las tres instrucciones de ip6tables.

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Se instala un servidor Nginx para el manejo de las peticiones web por puertos 80 o 443, aparcando en /var/www/html. Se verifica el estado del servicio con el segundo comando esperando un mensaje enabled.

```
aptitude install nginx
systemctl is-enabled nginx
```

Al igual que en Apache, debe de configurarse cada uno de los dominios a los que responderá nginx, por eso es necesario que se creara el archivo de configuración correspondiente. Además, este debe enlazarse a los dominios que están disponibles y activos. Se indican las dos instrucciones que hay que incluir en consola para que esto esté operativo.

```
nano /etc/nginx/sites-available/miescher.csic.edu.uy
sudo ln -s /etc/nginx/sites-available/miescher.csic.edu.uy /etc/nginx/sites-
enabled/miescher.csic.edu.uy
```

Cada uno de estos archivos debe contener la configuración de cada dominio, por lo que el contenido de miescher.csic.edu.uy debe ser el que se detalla a continuación. Por último, es necesario reiniciar el servicio para que los cambios surtan efecto.

```
server {
        # Escuchar en el puerto 80
        listen 80;
        # Root de la web
        root /var/www/html;
        # Orden de prioridad de los index
        index index.html index.html;
        # Nombre de domino
        server_name miescher.csic.edu.uy;
}
```

Una vez configurado el dominio, se obtiene un certificado SSL al dominio utilizando Certbot bajo la autoridad de Let's Encrypt, modificando automáticamente la configuración del servidor nginx para permitir la redirección de HTTP a HTTPS. Al finalizar la instalación de los paquetes es necesario reiniciar el servicio de nginx.

```
aptitude install certbot python-certbot-nginx
systemctl reload nginx
```

Certbot utiliza el complemento nginx para la obtención de los certificados SSL, activándose a través del primer comando a continuación. La segunda instrucción es quien genera el certificado y lo instala, y en la tercera instrucción se activa la renovación automática del mismo.

```
certbot --nginx
certbot certonly --nginx
certbot renew --dry-run
```

Las peticiones de aplicación python serán generadas dinámicamente a través de gunicorn. Para la implementación del webservice que se encarga de atender las peticiones de la aplicación en React se utilizó Flask, y para la generación del core del servicio, quien define las redes neuronales y el algoritmo genético se ha utilizado Pytorch. En los anexos se ofrece una breve reseña de cada una de las tecnologías utilizadas y la razón por las cuales se eligieron. En el artículo de Ellingwood y Juell (2019) se explica incluso más al detalle el proceso de configuración del entorno.

Es necesario comenzar instalando python y todas sus dependencias, incluyendo pip para la gestión de paquetes y python3-venv para la creación de entornos virtuales.

```
aptitude install python3-pip python3-dev build-essential libssl-dev libffi-dev
python3-setuptools python3-venv virtualenv python3-flask
```

La lógica detrás de estos servicios se puede ver en la imagen a continuación, donde se muestra gráficamente cómo interactúan las peticiones una vez que llegan al punto de entrada del servidor.

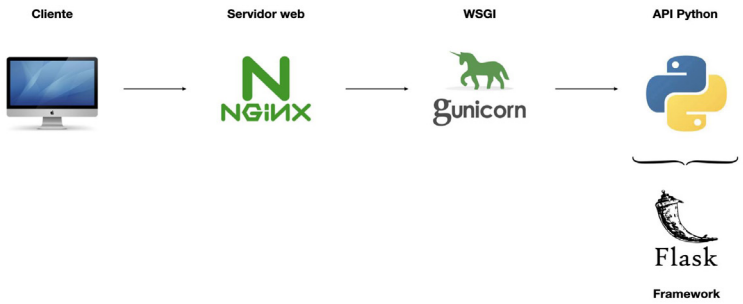


Figura 22: Interacción entre las partes de la API.

A partir de este punto se trabajará con el usuario gann. En la instalación, el directorio raíz de la aplicación flask se encuentra en /opt/api, por lo que es necesario crearlo. Además, es necesario crear un entorno virtual para el almacenamiento de una copia local de python y pip en ese directorio de pruebas. Finalmente se activa el entorno para empezar a trabajar en él, y en consola se indicará con un cambio en la línea de comando.

```
mkdir /opt/api
cd /opt/api
python3 -m venv apienv
source apienv/bin/activate
```

Una vez se está dentro del entorno virtual ya es posible instalar gunicorn y flask. Es necesario previamente agregar wheel para asegurar que los paquetes se instalan aunque falten archivos de wheel. Finalmente se procede a la instalación de gunicorn y flask.

```
pip3 install wheel
pip3 install gunicorn flask
```

Ahora bien, se crea un archivo como punto de entrada de la aplicación, que le indicará al servidor de Gunicorn cómo interactuar con la aplicación. El archivo se llama wsgi.py y su contenido se muestra en el siguiente código.

```
from api import app
if __name__ == “__main__”:
    app.run()
```

Se realizan las comprobaciones para asegurar que Gunicorn provee correctamente esta aplicación, pasándole el nombre del módulo que será el punto de entrada a la API más el elemento que ejecuta dentro de este. A través de consola puede verse la información del estado del servidor y desde el navegador, por el puerto 5000, se accede a la aplicación funcionando. El entorno virtual hasta aquí funciona correctamente, por lo que puede desactivarse utilizando el comando deactivate.

Se crea un archivo api.service en /etc/systemd/system para indicarle a Debian que debe iniciar automáticamente Gunicorn, haciendo funcionar con eso la API cada vez que el servidor inicie. Dentro de [Unit] definimos los metadatos y dependencias, mientras que dentro de [Service] se define el usuario, grupo que lo ejecuta, directorio de trabajo, se actualiza la PATH al ejecutable y los procesos que debe ejecutar.

Se agrega además [Install] que le indica al sistema operativo que deberá estar activo siempre que lo esté el sistema multiusuario, cargándose claro en el inicio. Luego de guardar, se inicia el servicio con sudo systemctl start api.

```
[Unit]
Description=Instancia de API usando Gunicorn
After=network.target

[Service]
User=gann
Group=www-data
WorkingDirectory=/opt/api
Environment="PATH=/opt/api/apienv/bin"
ExecStart=/usr/local/bin/gunicorn --workers 3 --timeout 600 --bind unix:api.sock -m 007
wsgi:app
StandardOutput=append:/opt/api/logs/stdout.log
StandardError=append:/opt/api/logs/stderr.log

[Install]
WantedBy=multi-user.target
```

Se reinicia el servicio de nginx y la aplicación queda montada y funcionando a través del puerto https.

Interfaz de comunicación de la API

Para la implementación de la API en Flask, se requieren los paquetes cors y jsonify. Cors (Cross-Origin Resource Sharing) es un paquete en python que prohíbe al servidor aceptar datos de distintos servidores para una misma petición, obligando a que todos los datos deban provenir de la misma fuente y asegurando que no se envíe código malicioso. Jsonify por el contrario, es un módulo que serializa información en formato JSON para poder ser enviada como respuesta a las peticiones del cliente.

```
pip3 install jsonify flask-cors
```

El código de la aplicación de base api.py, que tiene la responsabilidad de lanzar la API y definir las funciones a las que responde la misma, se muestra a continuación.

```
from flask import Flask,jsonify
from flask_cors import CORS # Para que se permita la política CORS
from utils import * # Importa el archivo con funciones auxiliares
import csv # importa las funciones para manipular archivos csv

app = Flask(__name__)
CORS(app) # Aplica la política de CORS sobre el objeto de la aplicación

# Definición de las funciones por caso de uso
@app.route('/')
def index():
    return jsonify({
        'Autor': 'Leroy Deniz',
        'Nombre': 'GANN',
        'Descripción': 'Servicio web para entrenamiento y optimización de RNAs'
    })

@app.route("/train", methods=["POST"], endpoint="train")
def api_train():
        # código del caso de uso Entrenamiento

@app.route("/test", methods=["POST"], endpoint="test")
def api_test():
        # código del caso de uso Evaluación

@app.route("/classify", methods=["POST"], endpoint="classify")
def api_classify():
        # código del caso de uso Clasificación
```

Cada cambio que se realiza en los archivos de flask requiere un reinicio del servicio para que estos sean tomados. Se adjunta el comando a ejecutar.

```
sudo systemctl restart api
```

Cada una de las funciones que tratará los casos de uso analizan los datos de entrada para su verificar su integridad antes de comenzar a realizar trabajo alguno. La recepción de las peticiones contra la API se hacen a través del método POST mientras que las peticiones procesadas son devueltas a través de GET. Desde Flask se recibe la información del formulario a través de request. form.get como se puede ver en la función debajo. Las verificaciones que realizará son respecto a la aceptación de términos y condiciones del servicio y la integridad del archivo csv o onnx. Para el caso del archivo csv, se utiliza request.file['train'], que permite traer el archivo serializado desde el formulario a través de la request. A continuación se adjunta el código de la función de train desde la puerta de entrada de la API a modo de ejemplo, los restantes casos de uso tienen sus propias funciones como se mencionó antes y son de similares características.

```
@app.route("/train", methods=["POST"])
def api_train():
    now = datetime.now()
    current_time = now.strftime("%H:%M:%S")
    print(f"\n[{current_time}]: Request(train)[{request.form}]")
    if request.method == "POST":

        # Recibir los datos desde el request
        pEmail = request.form.get('email')
        pServicio = request.form.get('servicio')
        pTerminos = request.form.get('terms')
        pTrain = request.files['train']
        pTrain_extension = request.form.get('train_extension')
        pTrain_filename = f"{pTrain.filename}.{pTrain_extension}"

        # verificar que acepta los términos del servicio
        if not pTerminos:
            return jsonify({'Error': 'Términos y condiciones no aceptados'})

         #Comprobar integridad de archivos
```

Se han definido las funciones que serán las puertas de entrada de la API, por lo que siguiendo la estructura sugerida por la documentación de Flask, se define el archivo utils.py que contendrá todas las funciones auxiliares que se utilizarán para la validación. Se adjunta un trozo del archivo como ejemplo de las funciones que contiene.

```
# Verificar que la extensión es csv
   elif not verificar_extension_csv(pTrain_filename):
      return jsonify({'Error': 'Formato no CSV'})

   # Deserializar el archivo recibido desde formulario
   pTrain = pTrain.read()
   print(f" -> Request(train)[{pTrain_filename}: {len(pTrain)}  bytes]")

   # Verificar que el archivo no está vacío
   if verificar_vacio_csv(pTrain):
      return jsonify({'Error': 'Archivo vacío'})
```

Una vez se han superado todos los filtros de verificación del fichero, se llama a la función de la API que se encarga del entrenamiento y evolución. En la función principal sigue este código:

```
# Si llega a este punto, está todo correcto
  else:
    try:
      pTrain = pTrain.decode('UTF-8')
      modelo, tst_scores, size_entrada = optimizar(pTrain)
      fichero = exportar_onnx(modelo, size_entrada)
      response = {
      "file": fichero.decode('latin1'),
        "error_perc": tst_scores[0],
        "num_neuronas": tst_scores[1],
        "num_capas": tst_scores[2],
        "avg_loss": tst_scores[3]
      }
      now = datetime.now()
      current_time = now.strftime("%H:%M:%S")
      print(f"[{current_time}]: Response(train)[{tst_scores} alongside File[{len(fichero)}
bytes]]\n")
      return response
    except Exception as e:
      now = datetime.now()
      current_time = now.strftime("%H:%M:%S")
      print(f"[{current_time}]: Exception(train)[{str(e)}]\n")
      return {"Error": str(e)}
```

Existe un problema con CORS a la hora de enviarle el archivo con el dataset, que se debía a un problema en la fase de Flask. React realiza la llamada por POST a través de AJAX a la API para pedir los headers, a fin de verificar si coinciden antes de enviarle la información. Como el servidor detectaba que el archivo a recibir era mayor al límite, devolvía directamente al cliente un error 413 avisándole que supera el límite permitido y acto seguido cerraba la conexión. Esto no alcanzaba a enviarle las cabeceras a modo de respuesta, por lo que es conveniente aumentar los límites de tamaño de archivo a recibir, de cara a trabajar con dataset de gran tamaño.

```
Access to XMLHttpRequest at 'https://miescher.csic.edu.uy/train' from origin 'https://gann.
es' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on
the requested resource.
```

El envío de datos al servidor presenta un problema a la hora del envío de archivos mayores a 2MB. Esto se debe a dos factores, en primer lugar y del lado del servidor, el tamaño máximo que Nginx permite por defecto son 2MB, además está la limitante de 16MB de Flask; por otro lado. Esto se corrige añadiendo una línea al archivo de configuración en /etc/nginx/nginx.conf con el tamaño máximo de archivos aceptados.

```
http {
    client_max_body_size 35M;
}
```

Para modificar el tamaño máximo de archivos en Flask se agrega en el código de la API la siguiente línea.

```
app.config['MAX_CONTENT_LENGTH'] = 35 * 1000 * 1000
```

Preprocesado de datos

EDA son las siglas de Exploratory Data Analysis, consiste en la revisión de los datos de entradas previo a cualquier tipo de entrenamiento o clasificación. Se aplica en el caso del presente trabajo sobre aquellos ficheros CSV sobre los que entrenará la red neuronal. En este paso se decide sobre una serie de cuestiones relativas a los datos que es conveniente enumerar y utilizar como checklist a la hora de ejecutar la etapa.[7]

- Cuantificación
    - ¿Cuántos registros hay?
    - ¿Son muy pocos para generalizar?
    - ¿Demasiados para poder procesarlos?
- Nulidad
    - ¿Hay campos nulos?
    - ¿Se anula todo el registro?
- Numeración
    - ¿Son valores discretos o continuos?
    - ¿Se puede obtener el tipo de dato? (text, int, float, etc.)
- Supervisión
    - ¿Cuál es la variable (columna) de clase?
    - ¿El dataset de train está balanceado?
    - ¿Hay suficientes muestras de cada clase y variedad para generalizar?
- Selección
    - ¿Hay variables que no aportan información?
    - ¿Pueden descartarse casos duplicados?
    - ¿Se tienen casos outliers? ¿Pueden eliminarse o deben conservarse? ¿Son errores de carga o datos reales?
- Verificación
    - ¿Están todas los los datos de las columnas unificados? (minúscula o mayúscula, puntos decimales o comas, singular o plural, etc.)
    - ¿Se tiene posible sesgo de datos?

El preprocessing o preprocesamiento de datos es, en Machine Learning, una etapa previa al entrenamiento de los modelos donde se realizan estas verificaciones y se tratan los casos que pudieran generar ruido en los resultados. Busca optimizar el juego de datos para corregir o descartar los casos poco fiables.

Para la manipulación de los ficheros que serán enviados en formato CSV desde la interfaz se utiliza la librería PANDAS, especializada en el manejo y análisis de estructuras. Toma todas las estructuras que se utilizaban en NumPy y las mejora, haciendo el manejo de los datos aún más fáciles ya que, por ejemplo, sus Arrays tienen índices por filas y columnas para poder recorrerlos. Permite lectura y escritura de archivos CSV como el que recibimos a través de la llamada a la API, entre otras muchas funcionalidades más. Es necesario instalar el paquete a través de pip3.

```
pip3 install pandas
```

El preprocessing se realiza desde el archivo utils.py, por lo que se importa la librería recién instalada. Para esta etapa se va a tener en cuenta qué política seguir frente a los valores faltantes, corruptos o inválidos. La función preprocessing dentro del fichero utils.py será quién manipule el conjunto de datos de ingreso de cara a este tratamiento, por eso recibirá el CSV como parámetro. Deberá deserializarlo, y aplicar la función read_csv de la librería pandas, quien transformará los datos en un dataset de elementos.

7. Aprendiendo Machine Learning (Bagnato, 2019)

```
import pandas as pd
import numpy as np
from scipy import stats
from io import StringIO

EPSILON = 0.00001

# Función para verificar que el archivo cumple la estructura de csv y tiene valores numéricos
def preprocessing(file):
  try:
    # Deserializar el archivo recibido desde formulario
    decFile = file.read().decode('UTF-8')
    # Convertir la información deserealizada de nuevo a un tipo File
    decFile = StringIO(decFile)
```

La política que se define para el tratamiento de estos datos, basándose en las preguntas anteriores y tomando en cuenta que el objetivo de este trabajo no es generar un preprocesado óptimo sino facilitar el juego de datos de cara al entrenamiento, se definen las siguientes características a ser implementadas en la función preprocessing del fichero utils.py.

- La primera fila corresponde al nombre de las variables

```
# Eliminar header=0 en caso que la primera fila no tenga los nombres de las columnas (depende de la política usada)
df = pd.read_csv(decFile, sep=',', header=0)
# Hay que cerrar el archivo
decFile.close()
```

- El dataset de train tendrá al menos 6 registros y no más de 100000.

```
# Verifica que el dataset tenga entre 6 y 100k rows
total_rows, _ = df.shape
if total_rows < 6 and total_rows > 100000:
    return False
```

- Elimina columnas cuyo dato sea el mismo en todas.

```
# Elimina todas las columnas que tengan el mismo valor, no aportan información y complejizan el modelo
df = df[[i for i in df if len(set(df[i]))>1]]
```

- Elimina columnas que tengan más de un 50% de datos vacíos.

```
# Elimina las columnas que tengan la mitad +1 posición a Null
min_nulls = (total_rows / 2) +1
df = df.dropna(axis=1, thresh=min_nulls)
```

- Se eliminan los registros donde existan campos vacíos o corruptos.

```
# Para aquellas filas que tengan valores a Null, se eliminan
df = df.dropna()
```

- Se eliminan los casos duplicados.

```
# Elimina los registros duplicados
df = df.drop_duplicates()
```

- Se eliminan aquellos registros outliers para evitar el ruido.

```
# Elimina los registros outliers, basándose en el z-score de la columna, en relación con la
media de la columna y la desviación estándar
constrains = df.select_dtypes(include=[np.number]).apply(lambda x:
  np.abs(np.abs(stats.zscore(x)) - 3) > EPSILON).all(axis=1)
df.drop(df.index[~constrains], inplace=True)
```

- Estandarizar los strings a lowercase.

```
# Estandarizar todos los strings a lowercase y convertirlos a valores numéricos
df = df.applymap(lambda s:s.lower() if type(s) == str else s)
columns = df.select_dtypes(include=[np.object])
for c in columns:
    df[c] = pd.factorize(df[c])[0]
```

- Randomizar las filas para atacar los datasets que vienen ordenados.

```
# Randomizar las filas del dataset para evitar datasets ordenados
df = df.sample(df.shape[0])
```

Para el tratamiento de datos faltantes se aplica una política de eliminación de líneas con los datos, en el entendido que es un error intentar llenar ese valor dado que con él se entrenará el clasificador. Se valora la falta del dato por sobre un dato erróneo.

Se divide el dataset en dos, por un lado aquellos datos que forman el conjunto de variables de decisión, determinantes a la hora de definir el modelo, y por otro un vector de tantas posiciones como filas tenga el dataset original; este último tendrá las variables de clases de los casos de muestra.

Implementación de las estructuras

Para la implementación del Core de la aplicación se utilizará Pytorch, una librería de Machine Learning que se basa en Torch. "Posee todas las funciones, clases y atributos necesarios para definir las redes neuronales, trabajar con los gradientes de las mismas, entrenarlas, validarlas, probarlas y actualizar sus pesos y sesgos. Además, también permite definir los tipos de datos con los que se trabajará, definir las funciones de activación, de optimización y de pérdida, obtener los parámetros de las redes, guardarlas, cargarlas y leer los datos [...]" (Belzunegui Gabilondo, 2020). Tiene incorporado a su ecosistema los conceptos de tensores y deep learning, que permite generar las estructuras de esta aplicación.

Las estructuras se establecen dentro del Core (/opt/api/core) y se crean en el fichero types.py, responsable de contener todos los tipos de datos que se requerirán para la implementación de la red neuronal y del algoritmo genético.

Hasta este punto se tiene instalado numpy y pandas, por lo que resta la instalación de Pytorch y cuda para poder continuar.

```
pip3 install pytorch torchvision python-pycuda
```

Podría pensarse en este módulo como un cúmulo enorme de operaciones algebraicas cuyo procesamiento requerirá una gran cantidad de tiempo. Para acelerar el tiempo se está utilizando CUDA, una plataforma de computación en paralelo, que realiza sus cálculos directamente en la GPU que, como es bastante conocido en esta disciplina, es un procesador que se especializa en realizar cálculos a un tiempo significativamente bajo. Se adjunta el código referente a las librerías importadas y el código de cuda.

```
import torch
import numpy as np
from torch import nn
from typing import List

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('Using {} device'.format(device))
```

Se comienza la definición de las estructuras que darán paso a la definición de las redes neuronales, así como del algoritmo genético. En primer lugar se definirá la estructura de la Capa, que establecerá los componentes de cara a la creación del gen. Los modelos se crearán utilizando los recursos que ofrece Pytorch, numpy y pandas.

Cada capa tendrá un identificador (nombre) que servirá para poder identificarla a la hora de graficar el modelo en futuras etapas, las neuronas de entrada y las de salida, que son estáticas frente a la evolución del modelo. La capa de neuronas de entrada (neuronas_entrada) se establece con la cantidad de variables de decisión del dataset de train; la capa de neuronas de salida (neuronas_salida) se define en función de la cantidad de datos distintos que pueda tomar la columna de variable de clase. Finalmente se define la función de activación de cada neurona (activacion), que por defecto será ReLU por su bajo costo computacional frente a otras de tipo sigmoidal o tangencial.

La expresión -> None hace referencia a que es un método y no una función. Una vez establecidos los parámetros de la función en su constructora, se deben inicializar para cada uno de los casos que se instancie.

Los pesos y los sesgos se inicializan aleatoriamente, por lo que se le establece una distribución uniforme como puede verse en las últimas dos líneas de la clase. Utilizando random.uniform propios de numpy, se le pasan los parámetros alto, bajo, tamaño, devolviendo las muestras aleatorias en una matriz numérica de tamaño fijo de (neuronas_entrada*neuronas_salida)

posiciones.

```
class Capa:

  ‘’’ Clase que representa una Capa ‘’’

  def __init__(
            self,
            nombre: str,
            neuronas_entrada: int,
            neuronas_salida: int,
            activacion: str = “relu” # función de activación por defecto
  ) -> None:
    ‘’’
    Esta clase permite definir en una abstracción a cada una de las capas de una RN
    Para instanciar una capa es necesario definirle los siguientes atributos:
      nombre (str): [description]
      neuronas_entrada (int): [description]
      neuronas_salida (int): [description]
      activacion (str, optional): [description]. Defaults to “relu”
    ‘’’
    # Inicializaciones de los parámetros de clase
    self.nombre = nombre
    self.neuronas_entrada = neuronas_entrada
    self.neuronas_salida = neuronas_salida
    self.activacion = activacion  # si es null, se mantiene relu

    # Toma los valores de entrada y salida e inicializa los pesos y el sesgo
    self.pesos = np.random.uniform(-1.0, 1.0, size=(neuronas_entrada, neuronas_salida))
    self.sesgos = np.random.uniform(-1.0, 1.0, size=(neuronas_salida, ))
```

Cada capa entonces define sus elementos constitutivos y todos los valores necesarios para su procesamiento. Una vez se tienen las capas definidas, lo siguiente es la implementación de la clase Gen. Cada gen es la definición de una red neuronal, por lo que va a estar formada por una capa de entrada, una de salida y al menos dos capas ocultas, condición necesaria para poder declararse como Red Neuronal Profunda. Hay que tomar en cuenta que, tanto la capa de entrada como la de salida, son casos particulares de la red, ya que son inmutables, la primera porque está definida en función de la cantidad de variables de decisión que no varían, mientras que la segunda porque está acotada en la cantidad de clases diferentes en los que puede ser clasificado un caso.

Para instanciar cada Gen de nuestro algoritmo, es necesario entonces pasarle a su constructora el número de neuronas de la capa de entrada, el de la capa de salida y una lista con la cantidad de neuronas que tendrá cada capa intermedias, entendiendo que cada elemento de esa lista corresponde a una de las capas ocultas, es decir, el número de capas viene dado por el cardinal de la lista. Luego podemos decir, por ejemplo, que de recibir [10, 8, 6, 4], se tendrán cuatro capas ocultas con 10, 8, 6 y 4 neuronas respectivamente.

Ahora bien, de cara a la utilización de este Gen en el algoritmo genético, es necesario inicializar la estructura de capas, por lo que a través de un for son añadidas cada una de estas creadas utilizando la clase Capa definida anteriormente, agregándose luego al modelo.

```
class Gen:

    ‘’’ Clase que representa un Gen ‘’’

    def __init__(
        self,  # indica que es un método
        neuronas_entrada:int,  # neuronas de entrada
        neuronas_salida:int,  # neuronas de salida
        capas_ocultas:List[int],  # capas con la cantidad de neuronas
    ) -> None:
        “””
        Constructora del Gen: instancia un objeto de la clase Gen que tendrá la definición
de una RNA

        Args:
        neuronas_entrada (int): número de neuronas en la capa de entrada
        neuronas_salida (int): número de neuronas en la capa de salida
        capas_ocultas (List[int]): lista con el número de neuronas en cada capa, cada
elemento representa además una capa oculta
        “””
        self.modelo = None
        self.capas: List[Capa] = []
        proxima_capa = neuronas_entrada

        for i, numero_neuronas in enumerate(capas_ocultas):
            self.capas.append(
                Capa(
                    nombre=f”Capa{i}”,
                    neuronas_entrada=proxima_capa,
                    neuronas_salida=numero_neuronas
                )
            )

            # la capa actual será la entrada de la próxima capa oculta
            proxima_capa = numero_neuronas

        # Caso particular para añadir capa de salida
        self.capas.append(
            Capa(
                nombre=”Salida”,  # Nombre de la última capa
                # número de neuronas de la capa inmediatamente anterior
                neuronas_entrada=proxima_capa,
                neuronas_salida=neuronas_salida,
                activacion=”sigmoid”  # función sigmode
            )
        )
```

Una vez se tiene la definición de la red neuronal en cada Gen, lo propio es utilizar el poder de Pytorch para poder generar el modelo de Red Neuronal propiamente dicho. Es por eso que se crea la clase NeuralNetwork que recibirá este Gen con la definición del modelo a crear.

Se comienza por definir partial_stack[] que llevará una lista de operaciones que definirán la red neuronal finalmente. Se recorre a través de un for las capas ocultas que se tratarán una a una (instancias de la clase Capa). A cada una de estas se le aplicará una transformación lineal utilizando nn.Linear de la librería Pytorch, dicha transformación no es otra que $y=xA^T+b$, es decir, almacenará en partial_stack una lista de operaciones que deberá realizar, (suma de los valores de entrada ponderados a través de los pesos), sumándole el sesgo a cada uno de los elemento.

Una vez tiene todos los nuevos pesos parciales, corresponde definir qué función de activación se le aplicará a cada una de esas capas, que por defecto, como bien se comentó antes, será ReLU por economía de procesos. La capa de salida tiene definida la función Softmax, cuyo costo computacional es más caro y por lo tanto se utilizará únicamente para este caso.

Finalmente, en la penúltima línea se le indica a Pytorch que el modelo final (self.stack), no es

otra cosa que la secuencia ordenada de todas las operaciones que se definieron en partial_stack, quién tendrá al final la lista con las operaciones ordenadas que procesarán cada una de las capas. Solo resta eliminar el contenido de la lista partial_stack que se dejará de utilizar al finalizar el for.

```
class NeuralNetwork(nn.Module):
    '''
        Clase que crea modelos de RN, utiliza Module, clase base provista por PyTorch
para la construcción de modelos de redes neuronales de una forma más simple
        Fuente: https://www.programmersought.com/article/36653677206/
    '''

    def __init__(self, gen: Gen):
        '''
            Se construye cada instancia de la clase Neural Network como la secuencia de
las operaciones que debe realizar a partir de una entrada, para alcanzar una salida
            Para eso requiere un Gen que defina la estructura de la red neuronal,
construyendo así el modelo
        '''
      super(NeuralNetwork, self).__init__()

      # Stack módulos
      partial_stack = []

      # Toma el número de capas y neuronas de cada una
      for capa in gen.capas:
         # Aplicar transformación lineal y agregar al módulo
         partial_stack.append(
             nn.Linear(capa.neuronas_entrada, capa.neuronas_salida))
         # Aplicar función de activación a todas las neuronas de la capa
          partial_stack.append(
           nn.ReLU() if capa.activacion == "relu" else nn.Softmax(dim=1)
          )

        # El iterador que construye el stack de módulos para luego ir a nn.Sequential,
almacenando las capas de manera secuencial
        self.stack = nn.Sequential(*partial_stack)

        # Bloque de código turbio, sólo se puede rezar aquí.
        i = 0
        def init_weights(m):
            nonlocal i
            if type(m)==nn.Linear:
                capa = gen.capas[i]
                m.weight = nn.Parameter(torch.Tensor(capa.pesos.T), requires_grad=True)
                m.bias = nn.Parameter(torch.Tensor(capa.sesgos), requires_grad=True)
                i += 1
        self.stack.apply(init_weights)

        del partial_stack
```

Se implementa además la función de propagación.

```
def forward(self, x):
    '''
    Define el cálculo forward del modelo, es decir, cómo devolver la
    salida del modelo requerida de acuerdo con el cálculo de entrada
    '''
    logits = self.stack(x.float())
    return logits
```

Se define ahora la función de train, quien se encargará del entrenamiento de la red neuronal, así como la función de test para verificar la bondad de cada nuevo clasificador.

```
def train(
    dataloader: DataLoader,
    model: NeuralNetwork,
    loss_fn: nn.CrossEntropyLoss,
    optimizer: torch.optim.SGD,
    device: str = "cpu"
) -> None:
    ''' Función de entrenamiento de la red neuronal '''
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

def test(
    dataloader: DataLoader,
    model: NeuralNetwork,
    loss_fn: nn.CrossEntropyLoss,
    device: str = "cpu"
) -> Tuple[float, float]:
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size

    return (100*correct, test_loss)
```

Para probar las funciones anteriores, creando una red neuronal y entrenándola utilizando un fichero csv previamente cargado, se utiliza la función que se detalla en el siguiente código. Para este ejemplo se entrena con el conjunto de datos Iris[8], que cuenta con 150 registros balanceados con casos de tres clases distintas.

En primer lugar se carga el contenido del CSV y se realiza el preprocessing de cara a una limpieza de los datos (ver apartado Pre procesado de datos). Una vez se tiene el dataset con el que se trabajará, es necesario mezclar los registros para evitar trabajar con un dataset ordenado; para esto se utiliza el método df.sample(df.shape[0]) que toma el dataset preprocesado, randomiza los registros y acto seguido separa la última columna del resto del dataset, ya que esta última indica la clase a pronosticar y el resto son las variables de decisión.

Como ya se comentó, el dataset Iris presenta 150 casos de muestra, balanceados de a 50 casos por tipo de flor. El principal problema de no randomizar el dataset, es que utilizando 66% de train y 33% de test, la red neuronal entrenará sobre casos de las dos primeras clases y al utilizar el resto para test, su valor de acierto será 0% ya que no es capaz de clasificar sobre una tercera clase desconocida para ella.

Se toman todas las variables de decisión para definir la cantidad de neuronas de entrada, con la instrucción dV.shape[1] y la cantidad de clases distintas que puede tomar los registros con la instrucción dC.nunique(dropna=True).

8. Más información del dataset Iris en https://es.wikipedia.org/wiki/Conjunto_de_datos_flor_iris

Utilizando torch.tensor se obtienen las matrices multidimensionales de un mismo tipo de dato, que permiten los cálculos a nivel de CPU o GPU. En este punto se alcanza a tener un dataset con las variables de decisión y otra de clases, por lo que se dividirá cada uno de estos en test y train según el porcentaje definido en p_train.

	Decisión	Clases
Train (dV)	0.8	0.2
Test (dC)	0.8	0.2

Los tensores creados deben ser deben ser cargados en objetos de tipo TensorDataset para poder ser tratados por pytorch y finalmente utilizando DataLoader para poder ser cargados por lotes.

Se define a través de nn.CrossEntropyLoss() la función de pérdida de entropía cruzada con esa instrucción y con torch.optim.SGD el descenso de gradiente estocástico como método de optimización.

Finalmente, hay que definir los epochs que son las veces que se aplicarán los algoritmos de backpropagation, la cantidad de veces que pasan los datos por la red neuronal; luego a través de un for de epoch veces, se llama a la función de train y de test, con los parámetros que estas requieren.

Hasta aquí el entrenamiento.

De esta función de prueba, cada etapa de procesado se refleja en una vista similar a la siguiente. Cabe destacar que el porcentaje correspondiente al Accuracy es perfecto, puesto que la red sobre la que se entrena es realmente profunda, con tres capas ocultas de 1000, 500 y 300 neuronas respectivamente, lo que hace que sea capaz de interpretar los casos demasiado bien, cayendo muy posiblemente en overfitting[9].

9. Overfitting: sobreajuste o sobreentrenamiento del clasificador, adaptándose perfectamente a los casos que ha visto pero no siendo capaz de clasificar bien aquellos que no conoce.

```
if __name__ == "__main__":
  d = None
  with open("flores.csv", "rb") as f:
    df = preprocessing(f)
    dV = df.iloc[:, :-1] # Variables de decisión
    dC = df.iloc[:, -1] # Variables de clase
    print("**** DATASET ****")
    print(dV)
    print(dC)
    print("**** TYPES ****")
    print(dV.dtypes)
    print(dC.dtypes)

    try:
      entrada, salida = dV.shape[1], dC.nunique(dropna=True)
      capas = [1000,500,300]
      g = ct.Gen(entrada, salida, capas)
      model = ct.NeuralNetwork(g).to(device)

      p_train = 0.80
      bs = 10

      train_values = torch.tensor(dV[:int((len(dV))*p_train)].values)
      train_target = torch.tensor(dC[:int((len(dC))*p_train)].values)
      test_values  = torch.tensor(dV[int((len(dV))*p_train):].values)
      test_target  = torch.tensor(dC[int((len(dC))*p_train):].values)

      train_tensor = TensorDataset(train_values, train_target)
      test_tensor = TensorDataset(test_values, test_target)

      train_dataloader = DataLoader(train_tensor, batch_size=bs)
      test_dataloader = DataLoader(test_tensor, batch_size=bs)

      loss_fn = nn.CrossEntropyLoss()
      optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

      epochs = 1000
      for t in range(epochs):
        print(f"Epoch {t+1}\n-------------------------------")
        train(train_dataloader, model, loss_fn, optimizer)
        test(test_dataloader, model, loss_fn)
      print("Done!")

      # Input to the model
      x = torch.randn(bs, 1, 4, requires_grad=True)
      torch_out = model(x)

      # Exportar el modelo
      torch.onnx.export(model,                    # model being run
        x,              # model input (or a tuple for multiple inputs)
        "modelo.onnx",  # where to save the model (can be a file or file-like object)
        export_params=True, # store the trained parameter weights inside the model file
        opset_version=10,      # the ONNX version to export the model to
        do_constant_folding=True,  # whether to execute constant folding for optimization
        input_names = ['input'],   # the model's input names
        output_names = ['output'], # the model's output names
        dynamic_axes={'input' : {0 : 'batch_size'},  # variable length axes
          'output' : {0 : 'batch_size'}})
  except Exception as e:
    print(e)
```

```
Epoch 1000
-------------------------------
loss: 0.585844  [    0/  119]
Test Error.
Accuracy: 100.0%, Avg loss: 0.570036
```

Implementación del Algoritmo Genético

El módulo que se utiliza para esta parte de la implementación es DEAP[10], es una librería especializada en algoritmos evolutivos, permitiendo así de una forma más rápida la implementación de esta metaheurística en el Core. puede instalarse desde los repositorios utilizando la siguiente instrucción:

```
pip3 install deap
```

La evolución de las redes neuronales que se entrenarán, podrán fluctuar dentro de cierto rango para evitar cualquier desbordamiento; para eso se definen las siguientes constantes en el archivo constants.py.

```
'''
DEFINICIÓN DE CONSTANTES
'''

# Rango de capas ocultas que puede tener una RN
RANGO_CAPAS = (4, 8) # Min 3 capas ocultas y 1 de salida (por eso 4)
# Rango de neuronas que puede tener cada capa de una RN
RANGO_NEURONAS = (10, 30)
# Probabilidad de mutación de sesgos
PROB_MUT_SESGOS = 0.5
# Probabilidad de mutación de pesos
PROB_MUT_PESOS = 0.5
# Neural Network
MAX_EPOCHS_NN = 10
# Probabilidad de cruce entre genes
PROB_CRUCE = 0.5
# Probabilidad de mutación de una neurona
PROB_MUT_NEURONA = 0.5
# Probabilidad de mutación de capa
PROB_MUT_CAPA = 0.5
# Tamaño de la población inicial
TAMANO_POBLACION = 10
# Máximo de generaciones
MAX_GENERACIONES = 10
# Semilla para fijar los valores generados
SEED = None
# Valor mínimo aceptable
EPSILON = 0.00001
# Porcentaje del dataset para train (1-P_TRAIN para test)
P_TRAIN = 0.80
# Tamaño del lote para entrenamiento
BATCH_SIZE = 10
```

Se toma como referencia para la implementación de algunos tipos de datos y los operadores, el desarrollo realizado en el proyecto DeepGProp[11] de Liñán Villafranca y Merelo Guervós (2021).

Los individuos en el algoritmo genético no son otra cosa que una RNA, luego el inicializador se hará desde la función inicializador_gen.

10. Puede verse la documentación en https://pypi.org/project/deap/
11. Repositorio del proyecto: https://github.com/lulivi/deep-g-prop

```
def inicializador_gen(
    clase_gen: Callable,
    capas_entrada_salida: Tuple[int, int],
    rango_num_neuronas: Tuple[int, int],
    rango_num_capas: Tuple[int, int]
) -> Any:
    '''
    Inicializa los individuos de manera uniforme (capas y número de neuronas)
    '''
    #  Inicializa una lista de capas ocultas de manera uniforme
    hidden_layers = np.random.randint(
        rango_num_neuronas[0],
        rango_num_neuronas[1] + 1,
        random.randint(*rango_num_capas),
    ).tolist()

    # Crea el individuo de la clase Gen, enviándole como parámetro la capa de entrada,
salida y las ocultas recién inicializadas
    return clase_gen(capas_entrada_salida[0], capas_entrada_salida[1], hidden_layers)
```

Se define además la caja de herramientas para el tratamiento de genéticos, donde hay que definir la función Fitness que será la que determine qué tan bueno es un gen y, además, sobre qué conjunto de datos se aplica, define también la función de creación de la población inicial.

```
def configure_toolbox(
    dataset: Tuple[DataLoader, DataLoader],
    nin_nout: Tuple[int, int],
    rango_neuronas: Tuple[int, int],
    rango_capas: Tuple[int, int],
    prob_mut_sesgos: float,
    prob_mut_pesos: float,
    max_epochs_ml: int
):
    '''
    Función de inicialización del algoritmo genético
    '''
    toolbox = base.Toolbox()
```

Esta toolbox del genético deberá tener una serie de definiciones establecidas para su utilización. A continuación se establecen las definiciones enumeradas.

- Define la población inicial del genético.

```
toolbox.register(
    “poblacion”,
    tools.initRepeat,
    list,
    toolbox.individuo
)
```

- Registra el individuo del algoritmo genético.

```
toolbox.register(
    “individuo”,
    inicializador_gen,
    creator.Individual,
    nin_nout,
    rango_neuronas,
    rango_capas
)
```

- Define cuál es la función de evaluación y sobre qué datasets se aplica.

```
toolbox.register(
    “evaluar”,
    evaluador_gen,
    trn=dataset[0],
    tst=dataset[1],
    max_epochs=max_epochs_ml
)
```

- Define el operador de reproducción: crossover.

```
toolbox.register(
    “cruzar”,
    operador_crossover
)
```

- Define la función de mutación de sesgos.

```
toolbox.register(
    “mutar_sesgos”,
    mutador_pesos_sesgos,
    attribute=”sesgos”,
    gen_prob=prob_mut_sesgos
)
```

- Define la función de mutación de pesos.

```
toolbox.register(
    “mutar_pesos”,
    mutador_pesos_sesgos,
    attribute=”pesos”,
    gen_prob=prob_mut_pesos,
)
```

- Define la función de mutación de neuronas.

```
toolbox.register(
    “mutar_neurona”,
    mutador_neurona
)
```

- Define la función de mutación de capas.

```
toolbox.register(
    “mutar_capa”,
    mutador_capa,
    rango_neuronas=rango_neuronas
)
```

Función fitness

Es necesario definir cómo actuará la función fitness, ya que como se indicó anteriormente, es una función de evaluación múltiple que evalúa cuatro métricas. En los comentarios puede verse el flujo que realiza el algoritmo de evaluación.

```
def evaluador_gen(
    gen: Gen,  # Gen individual a evaluar
    trn: DataLoader,  # Datos de Train
    tst: DataLoader,  # Datos de Test
    max_epochs: int # Número de epochs a realizar
) -> Tuple[float, int, int, float]:
    '''
    Evaluación de cada gen individual
    '''

    # Medición del tiempo
    tiempo_inicial = time.perf_counter()

    # Crea una lista con el número de neuronas de cada capa
    tamanos = [capa.neuronas_salida for capa in gen.capas]

    # Crea un modelo con la información del gen individual
    modelo = NeuralNetwork(gen).to(device)

    # Se define la función de pérdida y backpropagation para optimizar a través del cálculo
de las derivadas
    loss_fn = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(modelo.parameters(), lr=1e-3)

    epochs =max_epochs # número de veces que pasarán los datos por la RN
    acc, avg_loss = 0.0, 0.0  # valores iniciales de las métricas
    for t in range(epochs):
        #print(f"Epoch {t+1}\n-------------------------------")
        train(trn, modelo, loss_fn, optimizer, device)  # entrena
        acc, avg_loss = test(tst, modelo, loss_fn, device)  # testea

    # Para optimizar la RN, se calculan los tamaños: cantidad de capas y total de neuronas
    num_neuronas = sum(tamanos)
    num_capas = len(tamanos)

    gen.modelo = modelo

    # Calcula el error porcentual que presenta la RN
    error_perc = 100.00 - acc

    # Devuelve las métricas del entrenamiento actual
    return (error_perc, num_neuronas, num_capas, avg_loss)
```

Operadores de selección, reproducción y mutación

Se definen los operadores de selección, reproducción y mutación para el algoritmo genético, de forma que este pueda evolucionar las redes neuronales. Dada la cantidad de funciones y el tamaño de cada una de ellas, no se adjunta el código fuente a esta memoria sino hasta el final.

El operador de selección elegido es Elitista, el operador de reproducción elegido es Crossover de 1 punto. Finalmente se han definido cuatro operadores de mutación que actúan sobre las capas, las neuronas, pesos y sesgos de acuerdo a una probabilidad establecida anteriormente.

Para ver el código fuente de los operadores, puede hacerlo en el Anexo III.

Población inicial

La población inicial del algoritmo genético se inicializa en el fichero core/__init__.py a través de herramientas definidas en el toolbox. La cantidad de elementos está definida en constants.py, fijada inicialmente en 10 individuos.

```
population = toolbox.poblacion(n=tamano_poblacion)
```

Estructura del output

El output del core es la definición de una red neuronal en formato ONNX, siglas de Open Neural Network Exchange, siendo este un formato abierto para el intercambio de modelos de aprendizaje profundo en machine learning, un estándar.

Se puede acceder a la documentación del conversor de formato a través de su repositorio en GitHub en el siguiente enlace.

Se adjunta el fragmento de código de utils.py, donde se exporta el modelo onnx junto con las métricas del clasificador resultado de la función de test.

```
# Entrada del modelo
x = torch.randn(BATCH_SIZE, 1, entrada, requires_grad=True)
torch_out = opt(x)

# Exportar el modelo
fichero = ""
with torch.no_grad():
  with BytesIO() as f:
    torch.onnx.export(
      opt, # Modelo de ejecución
      x, # Entrada de modelo (o una tupla para múltiples entradas)
      f, # Dónde guardar el modelo (puede ser un archivo o un objeto similar a un
archivo)
      export_params=True, # Almacenar los pesos de los parámetros entrenados dentro del
archivo del modelo
      opset_version=10, # La versión ONNX para exportar el modelo a
      do_constant_folding=True, # Si ejecutar plegado constante para optimización
      input_names = ['input'], # Nombres de entrada del modelo
      output_names = ['output'], # Nombres de salida del modelo
      dynamic_axes={'input' : {0 : 'batch_size'}, # Ejes de longitud variable
      'output' : {0 : 'batch_size'}}
    )
    fichero = f.getvalue()
  return fichero, tst_scores
```

Capítulo VI

Diseño del Frontend

Diseño de Frontend

"Un error común que las personas cometen al tratar de diseñar algo completamente infalible es subestimar el ingenio de los tontos completos."

DOUGLAS ADAMS

Para la presentación y manipulación del sistema se desarrolla una interfaz web realizada que permite la utilización del servicio y el acceso a la documentación de uso, entre otras cuestiones asociadas a la herramienta.

Análisis de herramientas y alternativas disponibles

Para el diseño del frontend de la aplicación se analizó y se decidió por una tecnología de todas las disponibles en el mercado. A continuación se presenta un breve resumen de cada tecnología candidata para la implementación y diseño.

Candidata 1: .NET
Destaca su sencillez a la hora de desarrollar una aplicación en tres capas, simplificando su el proceso de diseño e implementación, además brinda una forma muy fácil y amigable de configurar los servicios web lo que facilita mucho el manejo de este tipo de soluciones. El problema encontrado son las dependencias a la hora de desplegar la aplicación en la nube, la cuota asignada por cuenta puede no ser suficiente para el tiempo y el uso que se le dará durante su desarrollo, lo que implica una inversión en términos de tecnología.

Candidata 2: PHP
Este lenguaje es muy flexible a la hora de desarrollar en él, entre otras cosas, permite una vinculación casi nativa con JavaScript, lo que permite la interacción vía AJAX en cualquier parte de la aplicación. Los métodos para consumir APIs no son particularmente intuitivos a la hora de programarlos. Esta herramienta tiene un punto a favor y es el conocimiento y uso por más de doce años desarrollando en ella, lo que facilita el proceso de manera significativa.

Candidata 3: ReactJS
Es un framework de JavaScript, que sigue el paradigma de Programación Orientada a Componentes y como tal, se basa en el encapsulamiento de código HTML, CSS y Javascript de tal forma que cada componente se vuelve independiente y reutilizable. Cada uno de estas piezas o componentes se implementa utilizando la sintaxis de JSX que embebe el código HTML dentro del propio JavaScript. A diferencia del paradigma clásico de desarrollo web, donde cada tipo de contenido se almacena en un directorio particular (css, js, img, etc), este framework permite un nivel de organización de código que se traduce en optimización y flexibilidad.. Tiene de manera nativa un DOM dinámico, lo que permite actualizar contenidos sin renderizar toda la página, ahorrando tiempo. React Native facilita la generación de aplicaciones iOS y Android con su propio código. El punto en contra de esta herramienta es que no fue vista durante el desarrollo del Grado.

Elección.
Finalmente, la perspectiva de aprender una tecnología tan popular como ReactJS e incluirla dentro del marco de un trabajo de fin de grado de ingeniería del software, fue quizá el punto clave de la decisión. Previo análisis en profundidad sobre la forma de trabajar con este framework, y habiendo asegurado que todas las funcionalidades que se requieren para el desarrollo de la aplicación pueden ser solventadas con esta herramienta, se construye el frontend sobre la versión de React JS 17.0.2.

Se utilizan además otras librerías que facilitan el maquetado y el negocio dentro de la aplicación. A continuación se indican las herramientas necesarias para su correcta compilación.

- Axios 0.21.1: es un framework de javascript que facilita las llamadas HTTP(S) a un servidor a través de unas pocas líneas en javascript.
- Bootstrap 5.0.2: es un framework CSS que permite el desarrollo de interfaces limpias y responsivas.
- Web Vitals 1.1.2: es una herramienta que permite medir la experiencia de usuario en un sitio web, bajo los algoritmos de posicionamiento de Google.

Sobre el framework ReactJS

Tomando la definición de Moreno Cantó (2018), React es una librería de código abierto desarrollada por Facebook que permite la construcción de interfaces gráficas, enfocándose sobre todo en Landing Pages o páginas de una sola vista. Su arquitectura se basa en el diseño de componentes que pueden reutilizarse una y otra vez a lo largo nuestro código, facilitando considerablemente el número de líneas de código frente a lo que sería el código lineal en HTML simple.

Una de las principales características de esta tecnología es la explotación de sus funcionalidades basadas en JavaScript, lo que hace que el código de integración con la API sea un copy and paste en comparación con otros frameworks que implica una generación específica. Por otro lado, Quintero Rodríguez (2016) la define como "una librería de código abierto de JavaScript para crear interfaces de usuario, esta librería es mantenida por Facebook, Instagram y la comunidad de desarrolladores JavaScript. Esta librería ofrece grandes beneficios en cuanto a rendimiento, modularidad, además, promueve un flujo muy claro de datos y eventos, facilitando así la planeación y desarrollo de aplicaciones complejas".

Además, sugiere la utilización de ReactJS para manejar vistas en el desarrollo de un modelo MVC, basándose en una optimización de la performance del DOM. React, a diferencia de otros frameworks basados en JavaScript utiliza el concepto de DOM Virtual, donde realiza los cambios antes de aplicar los cambios en los puntos de variabilidad, mejorando los tiempos de respuesta al acotar la cantidad de información a actualizar.

Como afirma Diaz Arteaga (2018), este DOM Virtual genera sub árboles de nodos a partir de estos cambios de estado para identificar las actualizaciones realizadas y mantener información entre las acciones del usuario.

Por las contribuciones que se han realizado a partir de los autores y a modo de definición integral, React es una herramienta que permite desarrollar sitios web de forma encapsulada por componentes, permitiendo su reutilización y velando por el rendimiento en términos de tiempo y datos.

React presenta una serie de ventajas frente a otros frameworks como la utilización de componentes declarativos y reutilizables, encapsulamiento y separación de preocupaciones. Posee además una API compacta para su generación y manejo, siendo incluso muy corta su curva de aprendizaje. Gestiona el renderizado del lado del servidor e incrementa la cohesión a través de los componentes (Cerón Giraldo, 2019). También plantea algunas desventajas a la hora de su integración con una arquitectura MVC, ya que puede no ser tan intuitivo a la hora de hacer las llamadas a los componentes externos a la API. El framework obliga a la utilización de una sintaxis propia, JSX, que no es más que código JavaScript con funciones propias y código HTML intercalado, sin embargo esto puede resultar un tanto dificultoso frente a la comodidad de la sintaxis de JS.

Para Paz Jáuregui (2017), esta es una tecnología que se ha ganado un lugar de gran popularidad por eficiencia en el desarrollo de proyectos de mediano y gran porte de manera continua, facilitando el manejo de las tareas y la organización del proceso de desarrollo.

Cada componente se extiende la clase Component, heredando entre otras cosas, la capacidad de manejo de estados y de reacción de tipo AJAX frente al servidor. La función render es la única obligatoria en cada Componente, responsable de imprimir en pantalla todo aquello que esté dentro, pudiendo intercalarse código JSX de forma de parametrizar, como se puede ver en el código de ejemplo en {this.props.name}, donde name es un atributo definido en el array props perteneciente al propio componente, this.

El atributo array props contiene los valores de entrada al componente, así como el atributo array state contiene los datos de uso interno, es decir, aquellos que pertenecen a los cambios de estados pero sin salir del propio componente.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hola {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name=”Taylor” />,
  document.getElementById(‘hello-example’)
);
```

El código de las vistas es declarativo, lo que hace más fácil de entender el funcionamiento y los procesos de testeo.

Acceso a la aplicación

El sitio web que permite la utilización de los web services se puede acceder a través de la URL:

https://gann.es

Actualmente la aplicación frontend se encuentra desplegada en un hosting compartido en la nube, que facilita su acceso a través de cualquier navegador en desktop o mobile.

Módulo de vinculación con la API

Para la implementación del módulo, se utiliza la funcionalidad ofrecida por Axios, que nos permite consumir un webservice a través de una función realmente simple de javascript. Sabemos que el resultado de esta petición será un archivo JSON, por lo que se prepara el código para esperarlo.

Es necesario previamente que el formulario que recibe el o los datasets y otros datos, reaccione de manera específica a una función en concreto, utilizando onSubmit como disparador:

```
<form onSubmit={this.onFormSubmit} id=”formServicios”>
```

No obstante, para la carga de cada archivo se utilizan eventos y funciones distintas, con el objetivo de optimizar el tiempo de espera entre la carga del fichero y el envío del formulario. Los archivos cuando son cargados, se guardan en variables de estado que serán enviadas al servidor para su procesamiento, llegando a la función a través del propio evento event. Además, estos son validados en su extensión para verificar sean .csv o .onnx según el caso que corresponda como primera medida de validación por posibles archivos inservibles; para esto se utiliza el atributo accept=”.csv,.onnx”.

```
// función que pega el contenido del dataset en la variable de estado
onTrainUpload = (event) => {
    this.setState({ train: event.target.files[0] });
    console.log(this.state);
};
```

Una vez cargados los archivos y presionado el botón de envío del formulario, se cargan las demás variables de estado pendientes a través de referencias y se adjuntan a la variable formData que será el paquete de información que se enviará para procesar a través de POST al servidor.

```
// referencias a los campos del formulario
emailRef = React.createRef();
termsRef = React.createRef();

// función que se ejecuta al enviar el formulario
onFormSubmit = (e) => {

    // actualizar las variables de estado con los valores del formulario
    this.setState({ email: this.emailRef.current.value });

    ...

    // Cargar el objeto
    formData.append(
        'email', this.emailRef.current.value
    );
```

Finalmente se establece la comunicación con la API a través de axios, indicando explícitamente en los headers que lo que se enviará contiene archivos (tal y como ocurre en HTML y PHP). Se le define una función de callback para establecer qué hacer en cada caso.

```
axios({
    method: "post",
    url: `${API_URL}/train`,
    data: formData,
    headers: {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Headers': 'Content-Type',
        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET',
        'Content-Type': 'multipart/form-data'
    },
})

.then(function (response) {
    // todo correcto, aquí el código de acción favorable
    console.log(response);
})
.catch(function (response) {
    // mostrar código de error
    console.log(response);
});
```

Este código se requiere para cada una de las llamadas a la API, en función de a qué caso de uso estamos referenciando. Es decir, hay tres casos de uso básicos: (1) Entrenar: cargar un dataset de train y devolver la definición óptima de la red neuronal, (2) Evaluar: cargar la definición de una RN o un dataset de train para clasificar uno de test, y devolver los datos de acierto de esa configuración de la red y (3) Clasificar: cargar la definición de la red o un dataset de train y un dataset de test sin clasificar, devolviendo el dataset clasificado.

El acceso a los servicios está disponible a través de tres componentes distintos, que se acceden a través de las urls /entrenar, /evaluar y /clasificar. Cada una de estas vistas difieren sobre todo en el .then de la llamada ajax. Todos los componentes cuentan con cuatro divs principales que contienen la estructura principal de cada vista:

- div-entrenar (o div-evaluar o div-clasificar), que tendrá el formulario responsable de cargar los ficheros y recoger el mail a donde se enviará el resultado
- div-loading, cuyo contenido es la vista de carga temporal mientras la API entrena la red neuronal.
- div-metricas, donde se mostrará el resultado de las métricas.
- div-error, en caso de que algo falle, se muestra este contenedor en lugar del anterior.

La transición se realiza a través de cambios de clases que tienen dentro las medidas de opacidad y tiempo de transición.

```
if (“Error” in response.data) {
   document.getElementById(“div-loading”).classList.remove(‘showing’);
   document.getElementById(“div-loading”).classList.add(‘not-showing’);
   document.getElementById(“div-error”).classList.remove(‘not-showing’);
   document.getElementById(“div-error”).classList.add(‘showing’);
   document.getElementById(“p-error”).innerHTML = response.data.Error;
} else {
```

Utilizando las nuevas actualizaciones de React, se definen previamente las variables en State dentro del constructor, para poder actualizarlas durante el proceso. Luego, en cada Round Button se agrega el evento onChange={this.handleChange} que lanza la función lambda que actualiza el estado.

```
handleChange = (e) => {
    this.setState({ opcion: e.target.value }, this.handleSubmit);
}
```

Para la descarga del archivo, sea el modelo de RNA en formato ONNX o la información clasificada en formato CSV, se establece el código JSX que permite forzar la descarga del fichero recibido por el webservice.

```
var fileContent = utf8.encode(response.data.file);
fileContent = iconv.encode(Buffer.from(response.data.file), ‘iso-8859-1’);
var archivo = new Blob([fileContent], { type: ‘text/onnx’ });
var csvURL = window.URL.createObjectURL(archivo);
var tempLink = document.createElement(‘a’);
tempLink.href = csvURL;
tempLink.setAttribute(‘download’, ‘model.onnx’);
tempLink.click();
```

El contenido de los tres componentes se muestra a continuación, cuyo estado por defecto es oculto a través de la clase not-showing.

```
<div id='div-loading' className="not-showing">
    <br />
    <h2>Entrenando...</h2>
    <img src={loading} width="400px;" alt="loading" id="img-loading" />
</div>


<div id='div-metricas' className="not-showing">
    <br />
    <h2>Métricas</h2>
    <p>
    <li key='MET01'>  Número de capas: <span id="num_capas"></span></li>
    <li key='MET02'>  Número de neuronas: <span id="num_neuronas">
</span></li>
    <li key='MET03'>  Tasa de acierto: <span id="accuracy"></span>%</li>
    <li key='MET04'>  Error: <span id="error_perc"></span>%</li>
    <li key='MET05'>  Media de pérdida: <span id="avg_loss"></span></li>
    </p>
</div>


<div id='div-error' className="not-showing" border="1">
    <h2>Error</h2>
    <p id="p-error"></p>
</div>
```

Árbol de navegación de los componentes

La aplicación, como desarrollo en React, se estructura en una serie de componentes, no siendo otra cosa que un fragmento de código HTML con JSX, siendo este una extensión del lenguaje JavaScript al que estamos acostumbrados. Entonces, los archivos se estructuran en función de su tipo dentro del árbol de navegación.

```
/gann
|_____ assets
|       |_____ css
|       |       |_____ App.css
|       |       |_____ index.css
|       |
|       |_____ img
|               |_____ logo.svg
|
|_____ components
|       |_____ ApiConnect.js
|       |_____ Index.js
|       |_____ Menu.js
|       |_____ Header.js
|       |_____ ...
|
|_____ docs
|       |_____ memoria.pdf
|       |_____ manual.pdf
|
|_____ App.js
|_____ index.js
|_____ Router.js
|_____ package.json
```

Interfaz gráfica

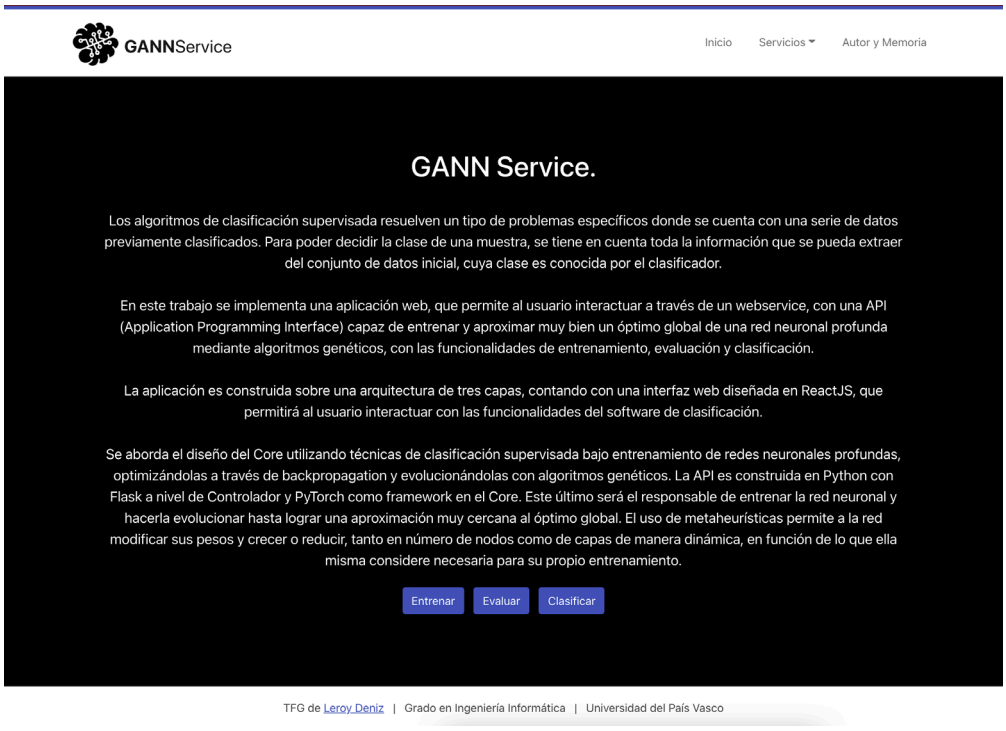


Figura 23: Interfaz gráfica de la aplicación

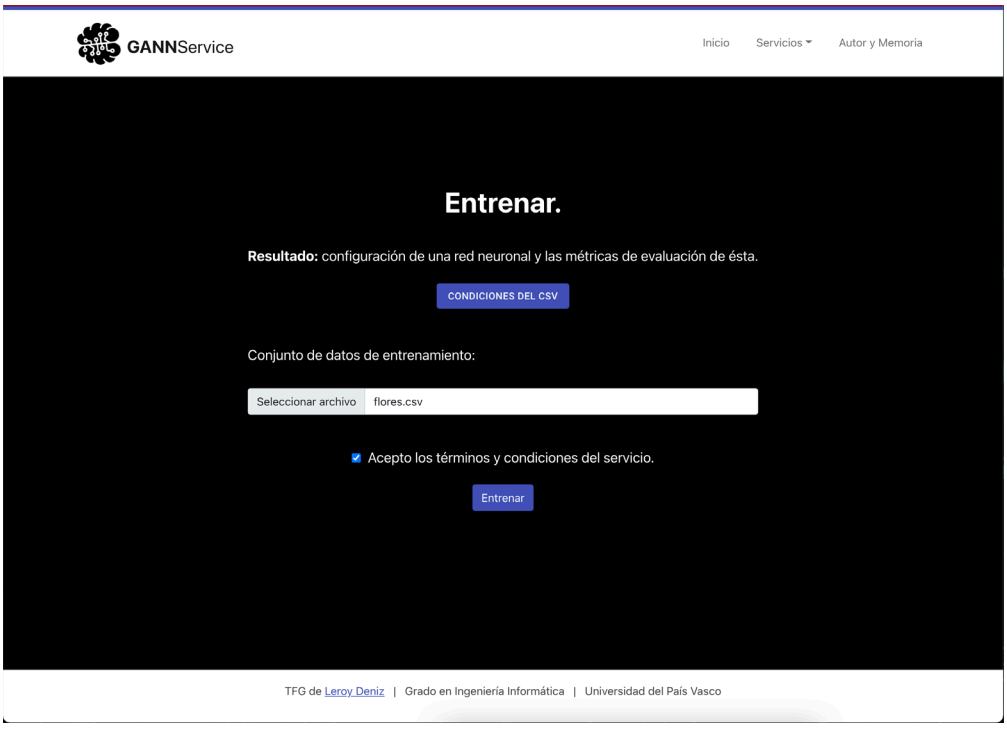


Figura 24: Interfaz gráfica del caso de uso Entrenar

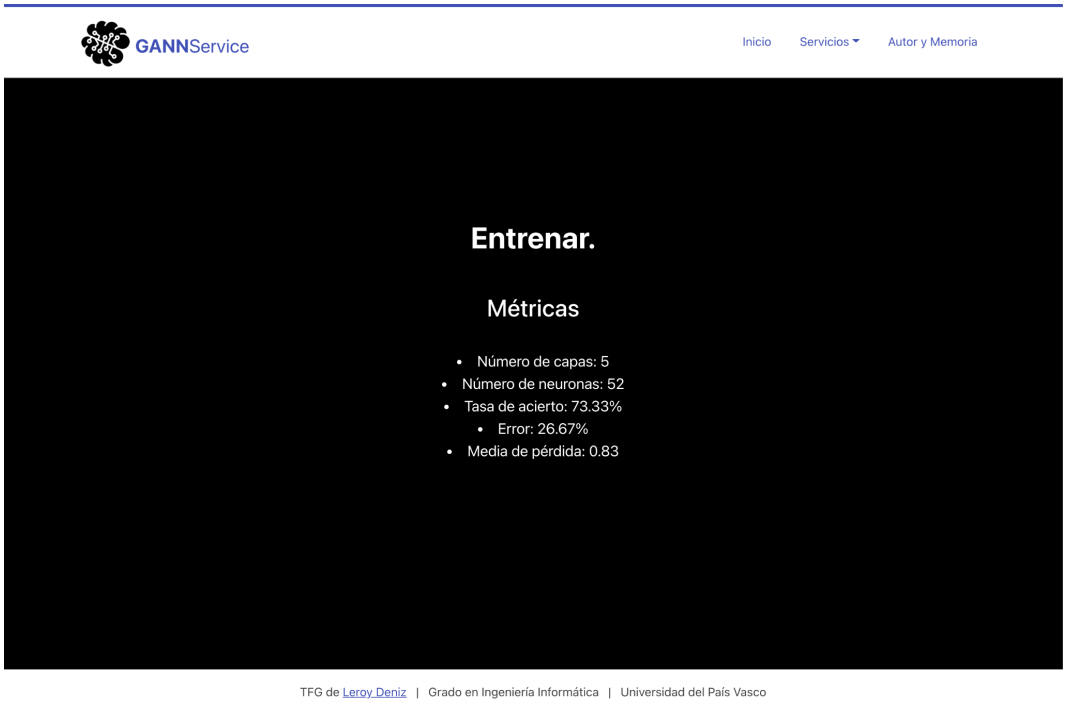


Figura 25: Interfaz gráfica de las métricas resultantes

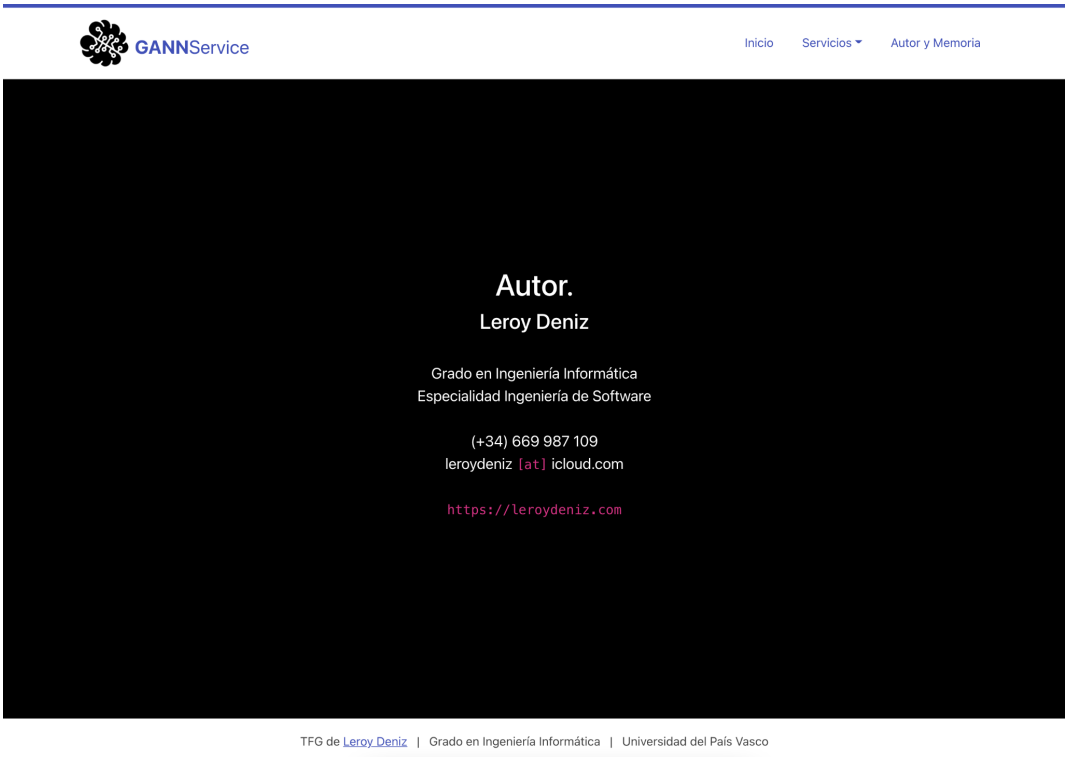


Figura 26: Interfaz gráfica de la vista de contacto

Capítulo VII

Análisis de calidad

Análisis de calidad

"Las pruebas pueden demostrar la presencia de defectos, pero no su ausencia."

EDSGER W. DIJKSTRA

Evaluación de un clasificador

Para este apartado se procede a realizar una serie de pruebas de los tres casos de uso de esta aplicación, ingresando un dataset de prueba y esperando recibir varios modelos evolucionados que permitan mostrar la variabilidad del sistema, así como su evaluación y las métricas que este presenta.

El dataset elegido será Iris[12], ya que contiene un número pequeño de registros y tres clases bien identificadas. Por otro lado, utilizaremos el dataset Clothes Size[13] que contiene 119.735 registros de tallas de ropa, cuyas variables predictoras son el peso, la edad y la altura de individuos. El clasificador está limitado en un máximo de 100 mil registros de entrenamiento, por lo que tomaremos el resto como datos para testear el caso de uso clasificación y ver qué tan buena es la red neuronal. Puede accederse a los datasets separados en train y test a través del siguiente enlace: https://bit.ly/3ygqD8T

	Iris			
Total casos dataset	150	150	150	150
RESULTADOS				
Número de capas	8	6	6	7
Número de neuronas	153	116	134	5185
Tasa de acierto	36.67	93.33	46.67	98.86
Error	63.33	6.67	53.33	1.14
Media de pérdida	1.18	0.62	1.08	0.47
CONDICIONES				
Rango de capas	4-8	4-8	4-8	4-8
Rango de neuronas	10-30	10-30	10-50	10-2000
Probabilidad mutación sesgos	0.5	0.5	0.5	0.3
Probabilidad mutación pesos	0.5	0.5	0.5	0.3
Probabilidad mutación capas	0.5	0.5	0.5	0.5
Probabilidad mutación neuronas	0.5	0.5	0.5	0.5
Probabilidad cruce	0.5	0.5	0.5	0.5
Epochs	10	10	10	1000
Tamaño población	2	10	4	8
Máximo generaciones	2	10	6	10

12. Dataset: https://es.wikipedia.org/wiki/Conjunto_de_datos_flor_iris
13. Dataset: https://www.kaggle.com/tourist55/clothessizeprediction

A partir de estos datos se puede ver el efecto de la aleatoriedad según sea la configuración elegida, así como la evolución de la red utilizando una metaheurística. Justamente la aplicación de estas técnicas es lo que no puede asegurar alcanzar un óptimo dado los recursos limitados del servidor, si se contara con la infraestructura necesaria podrían considerarse tomar constantes mayores que aumenten el área de búsqueda, yendo un poco más allá en la búsqueda de un óptimo global.

En clases de complejidad, este tipo de algoritmos son clasificados como clase NP-hard, ya que como problema de optimización, busca un óptimo global y para ello debería recorrer todo el espacio de búsqueda, lo que es computacionalmente imposible. Dado esto, de poder acelerar el tiempo de procesamiento, este tipo de problemas seguirán siendo muy difíciles pero podrían encontrarse mejores resultados aleatorios en el mismo tiempo dedicado.

En función del tiempo insumido para procesar un dataset de 150 registros, quedará para procesar y probar el dataset Clothes Size en cuanto se migre a una nueva infraestructura dedicada a machine learning, como por ejemplo la ofrecida por AWS[14].

Análisis del código

Se define una estrategia para la gestión de pruebas funcionales de la aplicación de cara a un análisis de comportamiento y fallos, que permitan solucionar los errores más complejos dentro del tiempo establecido para la implementación de este proyecto.

Por eso se utilizan dos enfoques distintos, uno puramente tecnológico a través de SonarCloud que ofrece métricas e identifica errores o mejoras posibles, y por otro se utiliza testing exploratorio que permite encontrar errores y mejoras en la medida que se va utilizando el propio software, pudiendo identificar los errores más claros y posibles soluciones a través del uso de herramientas de debug.

Reporte de SonarCloud

Sonarcloud es una plataforma de análisis de código online, capaz de estudiar grandes fuentes de código de proyectos y emitir reportes. Una de sus principales ventajas radica en el análisis continuo vinculado al repositorio GitHub del proyecto; con cada actualización de los ficheros en el repositorio, se lanza un nuevo estudio en la plataforma. Se realizaron los análisis correspondientes al código de la aplicación web y de la API, adjuntando a continuación los resultados. Ofrecen métricas para las correcciones del software y facilitan la detección de errores y malos hábitos de desarrollo.

14. Referencia: https://aws.amazon.com/es/sagemaker/

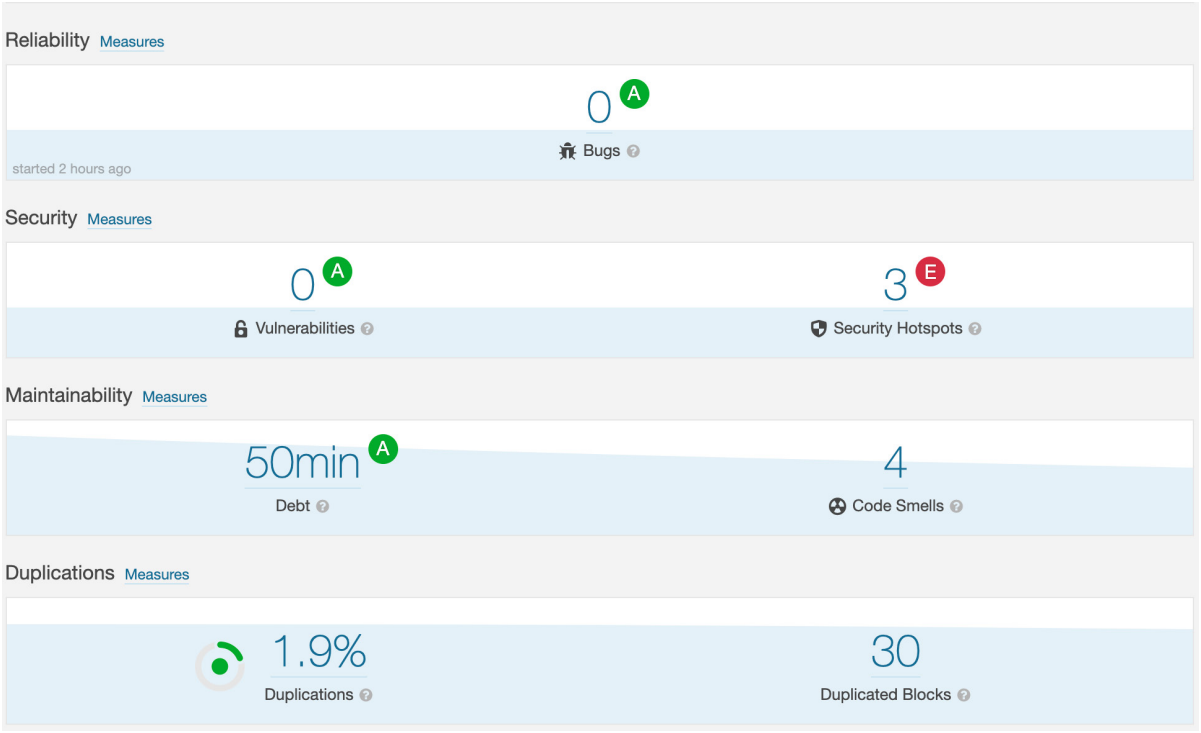


Figura 27: Reporte de SonarCloud sobre la aplicación web

El análisis nos refleja la existencia de varios Bad Smells, todos ellos de bajo impacto (dos ';' extras en el archivo CSS, algunos TODO que en realidad eran correctos,), y de fácil resolución, por lo que se procede a su corrección.

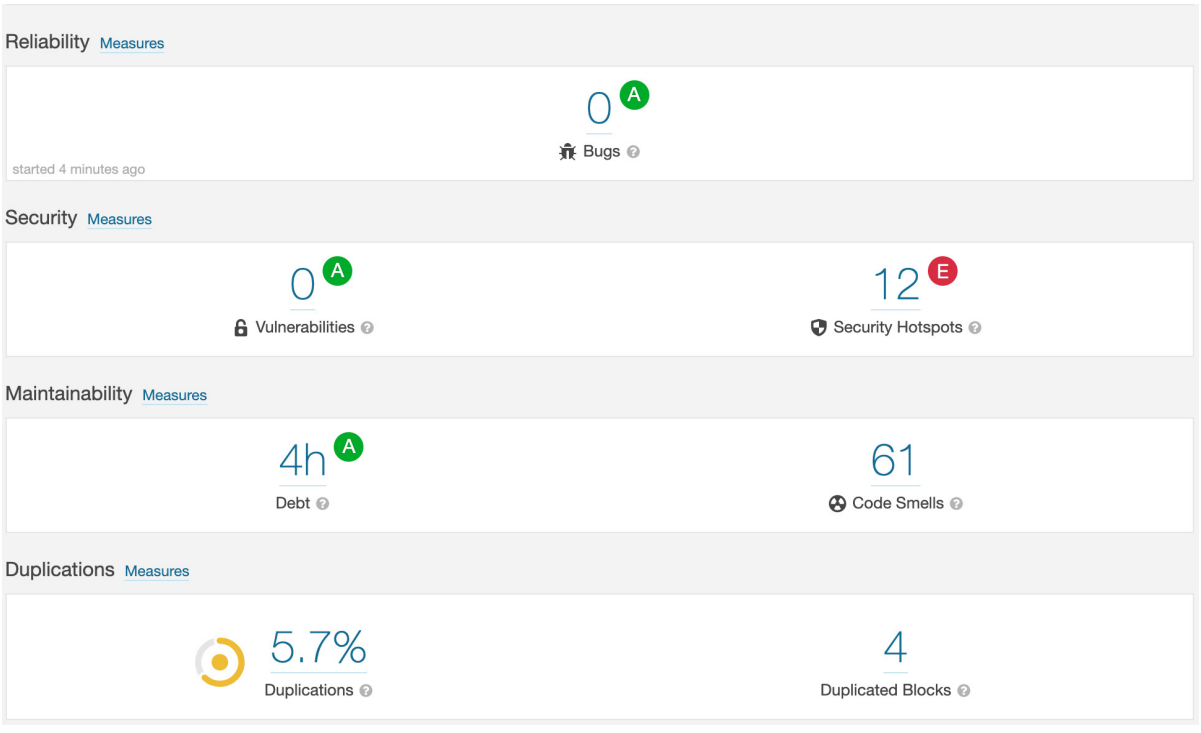


Figura 28: Reporte de SonarCloud sobre la API

El reporte de SonarCloud sobre la API en cambio, presenta Bad Smells de carácter críticos aunque en realidad pueden considerarse útiles a la hora de desarrollar. Importar todas las funciones de un fichero cuando efectivamente se utilizan, ahorra tiempo y permite acceder rápidamente a todas estas funcionalidades. Se han solventado la gran mayoría de los errores indicados en este reporte, además de proceder a las refactorizaciones necesarias para optimizar las líneas de código alivianando el peso del sistema.

Testing exploratorio

El testing exploratorio es una técnica de evaluación del software donde su característica principal es la simultaneidad del aprendizaje, diseño y ejecución. Es decir que todas las etapas coexisten y se realizan a la vez, entendiendo y corrigiendo el software, alcanzando un mejor diseño en las próximas pruebas, y todo esto realizado por la misma persona o equipos (Hourcade et al.). El testing exploratorio no es precisamente similar a una técnica de caja negra, puesto que en todo momento se conoce el funcionamiento interno del sistema, así como las posibles respuestas que debería tener.

Se aplica para este paradigma de testing en un intento de minimizar errores, construyendo una serie de pruebas iterativas donde cada etapa se ejecuta con las correcciones del paso inmediatamente anterior. Para esto se ha partido de un estudio de los tres casos de uso, tomando en cuenta las limitaciones que la infraestructura presenta.

Las vinculaciones del sistema por parte del usuario están dadas por la carga de ficheros, la descarga de los mismos, o las métricas que este muestra por la interfaz gráfica.

Caso de uso: Entrenamiento	**Resultado**	**Observaciones**
Fichero estándar con pocos registros	**CORRECTO**	Realiza preprocessing correctamente y
Fichero vacío	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero con más de cien mil registros	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero con cinco mil registros	**CORRECTO**	El tiempo de procesamiento es mayor a 10 minutos.
Fichero con cinco registros	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero con seis registros	**CORRECTO**	El sistema reconoce y procesa los registros
Términos y condiciones no aceptados	**CORRECTO**	El sistema verifica su aceptación a nivel de cliente y del servidor
Caso de uso: Evaluación	**Resultado**	**Observaciones**
Fichero onnx y el csv con menos de mil registros	**CORRECTO**	Se devuelven las métricas
Fichero onnx y el csv con más de cien mil registros	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero onnx y el csv vacío	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero csv para train y csv para test con 150 registros	**CORRECTO**	El sistema entrena y devuelve las métricas
Caso de uso: Clasificación	**Resultado**	**Observaciones**
Fichero onnx y el csv con menos de mil registros	**CORRECTO**	Se devuelven las métricas
Fichero onnx y el csv con más de cien mil registros	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero onnx y el csv vacío	**CORRECTO**	El sistema reconoce y devuelve el error
Fichero csv para train y csv para test con 150 registros	**CORRECTO**	El sistema entrena y devuelve las métricas

Los análisis anteriores han generado los resultados esperados, todos ellos devuelven errores capaces de identificar el problema.

Pruebas de performance

Los tiempo que insume la API para clasificar, como ya se explicó en apartados anteriores, son muy limitados a la configuración que se haya elegido para tratar cada red neuronal y el algoritmo genético, por lo que este no será un factor que se pueda tomar en cuenta aquí por dos razones fundamentales; en primer lugar porque el abanico y el tiempo de pruebas es exhaustivo para probar todas las posibles combinaciones entre los parámetros y en segundo lugar, porque está además en juego el efecto de la aleatoriedad que, aunque se fijen constantes, los mismos datos pueden dar dos soluciones distintas, lo que hace de esto un problema imposible de tratar.

Sin embargo, se han realizado las pruebas de performance utilizando la herramienta Google PageSpeed Insight, que ofrece una serie de métricas y posibles soluciones a los errores que detecta, además de medir la velocidad de carga y respuesta del servidor, pero no así el de su uso en acción.

Esta evaluación trabaja sobre los rangos 0-49 para un desarrollo pobre, 50 a 89 con sugerencias de mejora necesarias y 90-100 como una aplicación en buena salud.

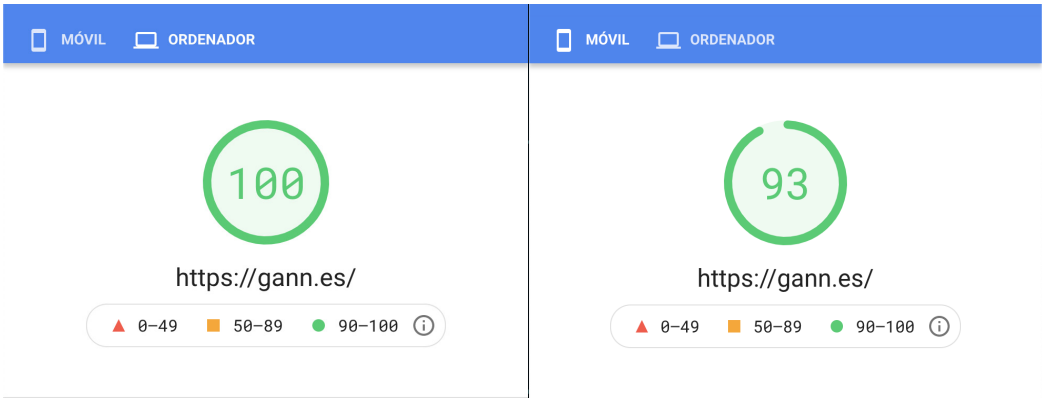


Figura 29: Resultados de los análisis de Google PageSpeed Insight

Como se puede ver, el nivel de performance es muy bueno y las sugerencias de mejora que ofrece son superficiales y no mejorarán de manera significativa el rendimiento de la aplicación. Se procede entonces a descartarlas y dar por cerrado el análisis de rendimiento y calidad en esta primera etapa.

Capítulo VIII

Seguimiento y control

Seguimiento y control

"Una meta sin un plan es solo un deseo."

ANTOINE DE SAINT-EXUPÉRY
El principito, 1943

En este apartado se describe brevemente la información importante respecto del seguimiento control del proyecto, haciendo especial énfasis en los aspectos de gestión: decisiones adoptadas y su justificación, desviaciones y riesgos incurridos en el proceso de desarrollo del proyecto, entre otra información relevante a la hora de ejecutar.

Desarrollo general del proyecto

El seguimiento del proyecto se ha dado de acuerdo al cronograma planteado a fines de junio. Pueden encontrarse en producción el servidor con su API en funcionamiento a través de https://miescher.csic.edu.uy y la aplicación web que la consume en https://gann.es.

Como forma de acceder rápidamente a la planilla de seguimiento de horas, puede hacerse a través del siguiente link: https://bit.ly/2UbdZZN.

Decisiones adoptadas

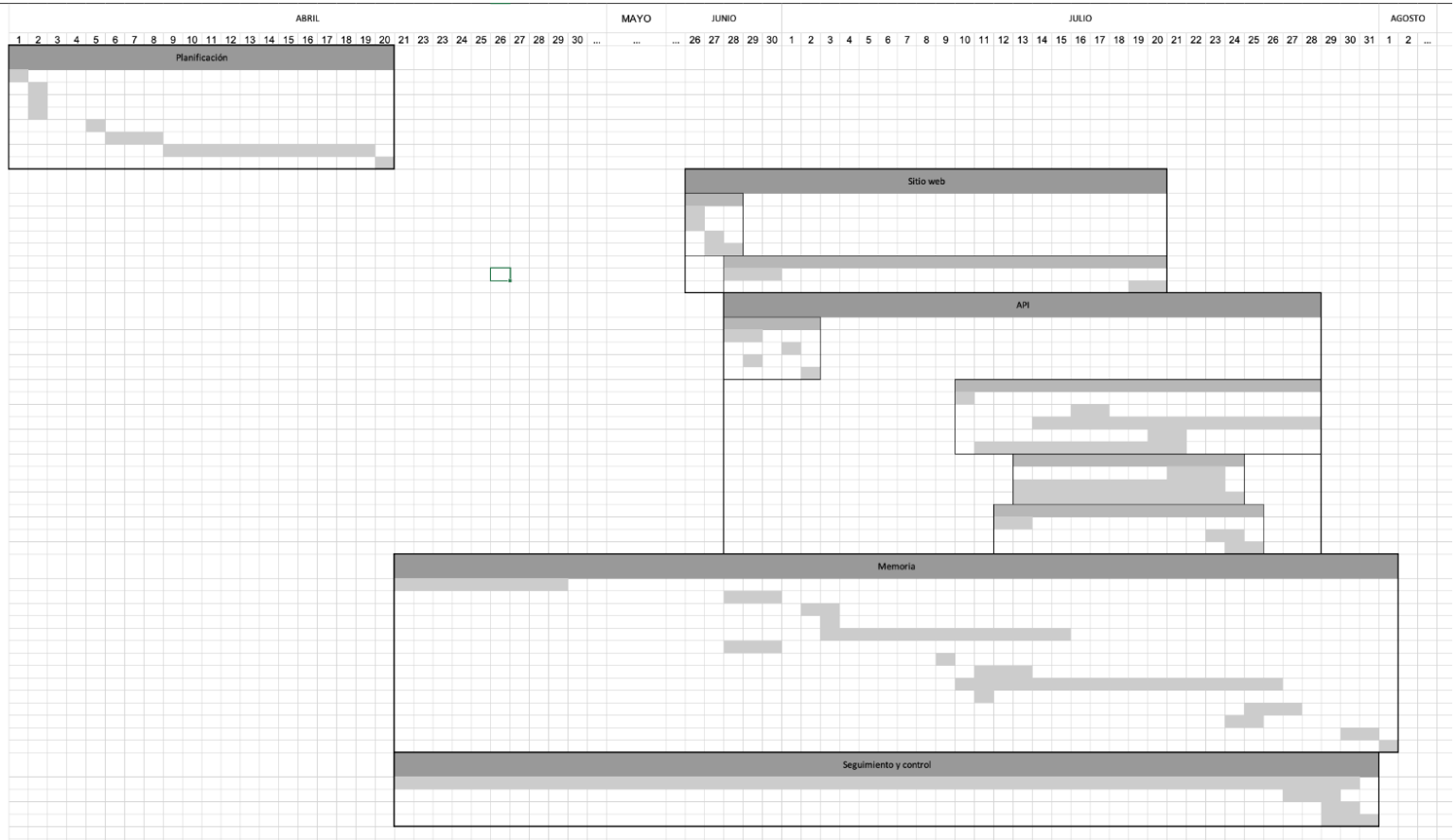
Como el formato estándar para exportar redes neuronales es ONNX (Open Neural Network Exchange), no es razonable exportar el resultado como XML, JSON o XML. Sin embargo, podrían exportarse las métricas resultantes del proceso de validation. Se fuerza la descarga del archivo onnx desde el navegador y se muestran a través de la interfaz los resultados obtenidos.

De todas formas y de cara a alcanzar el fin del proyecto en el tiempo establecido inicialmente, se añadirá como tarea en el apartado de trabajo futuro.

Entregables generados y accesos

1. Memoria del trabajo final.
2. Imagen del servidor web con la API en funcionamiento.
3. Comprimido con la aplicación web en React.

PLANIFICACIÓN DE TAREAS TFG			
PAQUETE/TAREA	**HORAS_ESTIMADAS**	**INICIO**	**FIN**
Planificación	**20**	**01-04-21**	**20-04-21**
Definición del tema	1	01-04-21	01-04-21
Definición de alcance	1	02-04-21	02-04-21
Definición de objetivos	3	02-04-21	02-04-21
Definición de resumen ejecutivo y abstract	2	02-04-21	02-04-21
Definición de cronograma macro y deadlines	1	05-04-21	05-04-21
Desarrollo de planificación general	2	06-04-21	08-04-21
Desglose de tareas y diagrama de Gantt	5	09-04-21	19-04-21
Entrega planificación inicial	5	20-04-21	20-04-21
Sitio web	**29**	**26-06-21**	**20-07-21**
Diseño, maquetación y contenido	**17**	**26-06-21**	**28-06-21**
Análisis de herramientas	1	26-06-21	26-06-21
Aprendizaje sintaxis de ReactJS	8	26-06-21	26-06-21
Preparar entorno de trabajo	1	27-06-21	27-06-21
Diseño de componentes y maquetación	7	27-06-21	28-06-21
Interfaz de comunicación webservice	**12**	**28-06-21**	**20-07-21**
Implementación módulo de vinculación	6	28-06-21	30-06-21
Ajustar vistas a lo que recibe de la API	6	19-07-21	20-07-21
API	**101**	**28-06-21**	**28-07-21**
Etapa de Diseño	**18**	**28-06-21**	**02-07-21**
Modelos de Casos de Uso	3	28-06-21	29-06-21
Modelo de Componentes	3	01-07-21	01-07-21
Flujo de eventos	5	29-06-21	29-06-21
Diagramas de Secuencia de Sistema	7	02-07-21	02-07-21
Etapa de Implementación	**63**	**10-07-21**	**28-07-21**
Implementación del entorno de producción	8	10-07-21	10-07-21
Implementación del módulo de red neuronal	20	16-07-21	17-07-21
Implementación del módulo de algoritmo genético	20	14-07-21	28-07-21
Preprocesado de datos de entrada	8	20-07-21	21-07-21
Implementación y verificación de webservices	7	11-07-21	21-07-21
Etapa de Entrenamiento	**8**	**13-07-21**	**24-07-21**
Definición del módulo de entrenamiento en API	3	21-07-21	23-07-21
Definición del módulo de clasificación en Controller	4	13-07-21	23-07-21
Establecimiento de JSON de retorno	1	13-07-21	24-07-21
Etapa de Verificación	**12**	**12-07-21**	**25-07-21**
Testeo del módulo de redes neuronales	4	12-07-21	13-07-21
Testeo del módulo de algoritmos genéticos	4	23-07-21	24-07-21
Análisis de datasets (workbench)	4	24-07-21	25-07-21
Memoria	**83**	**21-04-21**	**01-08-21**
Definición de estructura y contenido	3	21-04-21	29-04-21
Documentación de tecnologías utilizadas	6	28-06-21	30-06-21
Búsqueda de antecedentes	4	02-07-21	03-07-21
Planteo de Hipótesis	1	03-07-21	03-07-21
Contextualización y marco teórico	19	03-07-21	15-07-21
Descripción del desarrollo Frontend	8	28-06-21	30-06-21
Descripción de la configuración del entorno	8	09-07-21	09-07-21
Antecedentes de la aplicación	7	11-07-21	13-07-21
Descripción del desarrollo API	8	10-07-21	26-07-21
Estructura de datos de entrada	3	11-07-21	11-07-21
Conclusiones	10	25-07-21	27-07-21
Descripción del trabajo futuro	3	24-07-21	25-07-21
Diseño del póster	3	30-07-21	31-07-21
Entrega de la memoria	0	01-08-21	01-08-21
Seguimiento y control	**55**	**21-04-21**	**31-07-21**
Seguimiento del proyecto	14	21-04-21	30-07-21
Definición del plan de análisis de la calidad	5	27-07-21	29-07-21
Evaluación de la calidad	24	29-07-21	30-07-21
Correcciones resultantes del análisis de calidad	12	29-07-21	31-07-21



Desviaciones respecto de la planificación

A causa de la alta demanda de trabajos, exámenes y entregas requeridos por parte de las asignaturas pendientes durante los meses de mayo y junio, se ha dado el riesgo RI001 Planificación optimista, por lo que las tareas asignadas desde el 26/04/2021 han sido modificadas en su totalidad a realizarse en el período comprendido entre el 25/06/2021 al 31/07/2021. Si bien supone un retraso parcial en la realización de cada una de las tareas asignadas para el período, no supone gran cambio a nivel global ya que la planificación contemplaba tiempos libres entre paquetes.

No obstante, al 50% del proyecto realizado se cuenta con una desviación de 20 horas en contra, por eso se decidió la revisión de las tareas que pueden no ser indispensables para lograr el objetivo, quitando el paquete Lecciones Aprendidas liberando 10 horas del total. Esto posiciona el proyecto a 10 horas de desviación y aumenta el margen de maniobra para mantenerse dentro de las horas establecidas por requisito.

El tiempo de redacción del marco teórico insumió más de las horas previstas en un principio, lo que supuso una desviación de otras 6 horas frente a la planificación inicial. Sobre la finalización del proyecto se cuenta con unas 12 horas de desviación en contra, consumidas además aquellas pertenecientes al margen de contención. La vinculación entre la API y la aplicación ha llevado más horas de las estimadas inicialmente, lo cual debía ser previsible ya que se trabajó con Frameworks totalmente nuevos y, aunque la documentación de estos es verdaderamente buena, la vinculación entre todos es un caso demasiado particular como para poder encontrar ejemplos ilustrativos.

Tiempo dedicado y desviaciones

PAQUETE/TAREA	HORAS ESTIMA	HORAS REALES	INICIO	FIN	DESV
PLANIFICACIÓN	**20**	**20**	**01/04**	**20/04**	**0**
Definición del tema	1	1	01/04	01/04	0
Definición de alcance	1	1	02/04	02/04	0
Definición de objetivos	3	3	02/04	02/04	0
Definición de resumen ejecutivo y abstract	2	2	02/04	02/04	0
Definición de cronograma macro y deadlines	1	1	05/04	05/04	0
Desarrollo de planificación general	2	2	06/04	08/04	0
Desglose de tareas y diagrama de Gantt	5	5	09/04	19/04	0
Entrega planificación inicial	5	5	20/04	20/04	0
SITIO WEB	**29**	**50**	**26/06**	**20/07**	**-21**
Análisis de herramientas	1	3	26/06	26/06	-2
Aprendizaje sintaxis de ReactJS	8	8	26/06	26/06	0
Preparar entorno de trabajo	1	3	27/06	27/06	-2
Diseño de componentes y maquetación	7	10	27/06	28/06	-3
Implementación módulo de vinculación	6	12	28/06	30/06	-6
Ajustar vistas a lo que recibe de la API	6	14	20/07	20/07	-8
API	**101**	**89**	**28/06**	**30/07**	**+12**
Modelos de Casos de Uso	3	3	28/06	29/06	0

Modelo de Componentes	3	3	01/07	01/07	0
Flujo de eventos	5	4	29/06	29/06	+1
Diagramas de Secuencia de Sistema	7	5	02/07	02/07	+2
Implementación del entorno de producción	8	8	10/07	10/07	0
Implementación del módulo de red neuronal	20	8	16/07	17/07	+12
Implementación del módulo de algoritmo genético	20	17	14/07	28/07	+3
Preprocesado de datos de entrada	8	13	20/07	21/07	-5
Implementación y verificación de webservices	7	15	11/07	21/07	-8
Definición del módulo de entrenamiento en API	3	2	21/07	23/07	+1
Definición del módulo de clasificación en Controller	4	3	13/07	23/07	+1
Establecimiento de JSON de retorno	1	1	13/07	24/07	0
Testeo del módulo de redes neuronales	4	3	12/07	13/07	+1
Testeo del módulo de algoritmos genéticos	4	2	23/07	24/07	+2
Análisis de datasets (workbench)	4	2	30/07	30/07	+2
MEMORIA	**83**	**91**	**21/04**	**01/08**	**-8**
Definición de estructura y contenido	3	6	24/04	29/04	-3
Documentación de tecnologías utilizadas	6	5	28/06	30/06	1
Búsqueda de antecedentes	4	5	02/07	02/07	-1
Planteo de Hipótesis	1	1	03/07	03/07	0
Contextualización y marco teórico	19	25	03/07	03/07	-6
Descripción del desarrollo Frontend	8	9	28/06	28/06	-1
Descripción de la configuración del entorno	8	5	09/07	09/07	3
Antecedentes de la aplicación	7	8	11/07	13/07	-1
Descripción del desarrollo API	8	14	10/07	26/07	-6
Estructura de datos de entrada	3	2	11/07	11/07	1
Conclusiones	10	6	25/07	27/07	4
Descripción del trabajo futuro	3	2	24/07	05/07	1
Diseño del póster	3	3	30/07	31/07	0
Entrega de la memoria	0	0	01/08	01/08	0
SEGUIMIENTO Y CONTROL	**55**	**50**	**21/04**	**31/07**	**+5**
Seguimiento del proyecto	14	11	21/04	30/07	3
Definición del plan de análisis de la calidad	5	1	27/07	29/07	4
Evaluación de la calidad	24	24	2907	30/07	0
Correcciones resultantes del análisis de calidad	12	14	29/07	31/07	-2
TOTAL	**288**	**300**	**01/04**	**01/08**	**-12**

Riesgos confirmados y contingencia

Una cuestión que no fue tomada en cuenta en la planificación por motivos de desconocimiento de las herramientas, es la poca documentación de vinculación entre React y una API Restful a través de webservices, generando un aumento en el tiempo que se insumió a la hora de lograr generar el vínculo entre ambas. Esta vinculación entre ambos elementos del ecosistema genera problemas a la hora de enviar información superior a 2MB, lo que automáticamente se desata un problema en la política de CORS del servidor, ya que la información está llegando de a partes. La poca documentación y la inexperiencia derivó en un tiempo insumido mayor en la planificación.

Al 95% del proyecto avanzado, se habían superado las horas de contención, por lo que la solución era, o realizar un ajuste en el alcance y acortar ciertas funcionalidades, o bien agilizar en el desarrollo de las tareas restantes a fin de liberar tiempo. Se procedió a realizar un análisis de las tareas pendientes para determinar aquellas que podían ajustarse el tiempo de dedicación.

El motivo de la estimación incorrecta se basa principalmente en que se ha confirmado el riesgo RI002, habiendo tareas no contempladas dentro del proyecto inicial. Si bien estas tareas individuales no son significativas en términos de tiempo, la totalidad de ellas afectaron directamente al cronograma intermedio pero sin mover los hitos establecidos.

Capítulo IX

Conclusiones y trabajo futuro

Conclusiones

"La utopía está en el horizonte. Me acerco dos pasos, ella se aleja dos pasos. Camino diez pasos, ella se mueve diez pasos más allá. Entonces, ¿para qué sirve la utopía? Para eso, sirve para seguir caminando."

FERNANDO BIRRI POR EDUARDO GALEANO
Universidad de Cartagena de Indias, 1992.

Este documento aborda el desarrollo de un proyecto que sirva de soporte para el montaje y configuración de un sistema que permite entrenar redes neuronales profundas y las evoluciona a través de algoritmos genéticos. La solución planteada, pone en producción una aplicación web junto con la API que, utilizando una serie de frameworks -no vistos durante los cursos-, es capaz de resolver problemas de clasificación supervisada de manera satisfactoria.

Hasta la fecha no se ha encontrado una aplicación similar que permita generar y obtener un modelo de red neuronal desde la web, sino que permiten el entrenamiento y una configuración pero ninguna devolviendo un modelo utilizando el estándar ONNX. Esto resuelve un problema de portabilidad de modelos de machine learning pudiendo, de cara a la continuación del mismo en el futuro, configurar los parámetros necesarios para poder aproximar aún mejor a la solución óptima del modelo.

En cuanto a los modelos resultantes, el tiempo que insume el entrenamiento de una familia de redes neuronales, su evaluación en términos de métricas, selección, reproducción y mutación es significativamente alto cuanto más capas, neuronas y epochs presenta, por lo que es importante a los efectos de las pruebas realizadas o en pro de una puesta en producción para su uso, plantearse su implantación en un ecosistema dedicado a machine learning que permita trabajar con cálculos a una velocidad meyor.

Es por eso que se han utilizado un número bajo de todas las constantes, para así poder demostrar el funcionamiento sin prestar especial atención a la eficiencia de la propia red neuronal resultante en la versión de producción, aunque durante el proceso de análisis de calidad se han utilizado sí redes de varias capas ocultas y no menos de 500 neuronas por cada una de ellas, aumentando también en número de epochs para mejorar la precisión.

El tiempo consumido es demasiado alto en el entorno en el que está implementada la API, pero de contar con uno especializado, podría incluso aumentarse las estructuras en su versión publicada. Sin embargo, la performance para redes pequeñas es muy buena y en cuestión de segundos se alcanza un modelo que, sin ser muy preciso completamente, mejora por mucho la aleatoriedad.

La curva de aprendizaje de cada framework utilizado no es de menospreciar, puesto que para el desarrollar este proyecto se han utilizado todas herramientas nuevas que no habían sido vistas en el curso, que por un lado representan un desafío a la hora de generar los productos y por otro un riesgo importante de cara a cumplir con los plazos. La mitigación de este riesgo se alcanzó estudiando y definiendo bien la casuística del proyecto, intentando encauzar en todo momento las dedicaciones.

También es cierto que para su desarrollo no se han contabilizado las horas de aprendizaje de todas estas nuevas herramientas, que además ha representado un tiempo significativo, en el entendido

que este proyecto tiene un final definido, mientras que el aprendizaje de nuevas tecnologías es un activo que trasciende al cumplimiento de esta tarea y del cronograma aquí presentado.

No obstante, creo que lo más importante a destacar son las numerosas líneas de conocimiento abiertas que han ido resultando del proceso de aprendizaje y formación, siendo estas merecedoras de una profundización de cara a una continuación en una futura línea de investigación personal.

Trabajo futuro

"Y cuando creíamos que teníamos todas las respuestas, de pronto, cambiaron todas las preguntas."

MARIO BENEDETTI

A lo largo de este proyecto se han ido identificando una serie de elementos que se dan por supuesto a la hora de la implementación y del uso de la aplicación, por lo que de cara a una generalización del sistema, una mejora sería la posibilidad de la configuración por parte del usuario de todos los aspectos que serán condicionantes en el entrenamiento, filtrado o evaluación. Desde la propia interfaz web podría incorporarse los componentes necesarios para elegir qué tipo de validation realizar de cara a evaluar la bondad de la red neuronal, forzar el tipo de dato que se asocia a cada columna del dataset original, incluso permitir una definición de la política de preprocessing terciarizando en el usuario la responsabilidad de decisión sobre qué datos incluye y qué criterio prefiere.

Otro aspecto fundamental para permitir al usuario definir características, radica la posibilidad de elegir las constantes o parámetros tanto de las redes neuronales como del algoritmo genético, permitiendo así decidir el rango de capas ocultas que tendrán las redes, el número de neuronas mínimo y máximo de cada una de estas, las probabilidades de reproducción y mutación, entre otras. Esto asegurará al usuario tener más control sobre el modelo de red neuronal que obtendrá finalizado el proceso.

Podría implementarse en paralelo otra línea de mejora, permitiendo la elección de la topología de red neuronal que se prefiera, permitiendo el procesamiento de otros tipos de ficheros y, por ejemplo, el reconocimiento de patrones en imágenes o sonido utilizando redes neuronales convolucionales.

El software actual permite la clasificación supervisada en función del dataset que se ingresa, lo que permitiría una línea de mejora de cara a poder utilizar regresión para poder predecir valores en aquellas variables continuas.

Se ha derivado a este apartado la exportación de los datos de la red en formatos XML, JSON y CSV, quedando como tareas pendientes de la planificación inicial, así como el envío de los modelos y datasets clasificados por email a través de la API, ya que el esfuerzo de su implementación en el tiempo que se tiene es ampliamente superior a la disponibilidad.

Capítulo X

Bibliografía

Bibliografía

"Que otros se jacten de las páginas que han escrito;
a mi me enorgullecen las que he leído."

JORGE LUIS BORGES

1. Alonso Romero, L. (s. f.). Redes Neuronales. https://bit.ly/3dDnCHr
2. Bagnato, J. I. (2019, diciembre 12). Análisis Exploratorio de Datos con Pandas en Python [Blog]. Aprende Machine Learning. https://www.aprendemachinelearning.com/analisis-exploratorio-de-datos-pandas-python/
3. Arranz de la Peña, J. y Parra Tuyol, A. (s. f.). Algoritmos Genéticos. Universidad Carlos III de Madrid. Recuperado 5 de julio de 2021, de http://www.it.uc3m.es/~jvillena/irc/practicas/06-07/05.pdf
4. Belzunegui Gabilondo, I. (2020). Diseño e implementación de una librería de neuro-evolución para aprendizaje profundo con Pytorch [Tesis de grado, Universidad del País Vasco, Facultad de Informática]. http://hdl.handle.net/10810/48820
5. Besogain Olabe, X. (s. f.). Redes Neuronales Artificiales y sus aplicaciones. Dpto. Ingeniería de Sistemas y Automática, Escuela de Ingeniería de Bilbao, EHU. Recuperado 5 de mayo de 2021, de https://ocw.ehu.eus/pluginfile.php/40137/mod_resource/content/1/redes_neuro/contenidos/pdf/libro-del-curso.pdf
6. Calegari, D. (2002). Algoritmos Genéticos aplicados al Diseño de una Red de Comunicaciones Confiable [Tesis de grado, Universidad de la República, Facultad de Ingeniería]. http://hdl.handle.net/20.500.12008/3043
7. Calle Cárdenas, J. (2014). Diseño de un Algoritmo para. Evolucionar Redes Neuronales. Artificiales mediante Algoritmos [Tesis de grado, Universidad Católica de Santa María]. https://core.ac.uk/download/pdf/198129927.pdf
8. Cancela, H., Robledo, F. y Gerardo Rubino. (2003, octubre 3). Network design with node connectivity constraints. LANC '03: Proceedings of the 2003 IFIP/ACM Latin America conference on Towards a Latin American agenda for network research, La Paz, Bolivia. https://doi.org/10.1145/1035662.1035664
9. Castillo Valdivieso, P. A., Merelo, J. J., Prieto, A., Rivas, V. y Romero López, G. (2000). G-Prop: Global optimization of multilayer perceptrons using GAs. Neurocomputing, 35, 15. https://doi.org/10.1016/S0925-2312(00)00302-7
10. Ceron Galindo, J. M. (2019). React js: La nueva tendencia en aplicaciones web, enfocadas en el control dinámico de datos [Monografía, Universidad Cooperativa de Colombia]. http://hdl.handle.net/20.500.12494/14314
11. Díaz Arteaga, M. (2018). Simulador didáctico de una arquitectura de planificación estática [Tesis de grado, Universidad de la Laguna]. http://riull.ull.es/xmlui/handle/915/12393
12. Ellingwood, J. y Juell, K. (2019, diciembre 5). Cómo preparar aplicaciones de Flask con Gunicorn y Nginx en Ubuntu 18.04 [DigitalOcean.com]. Developers supporting developers. https://do.co/3wulxEd
13. Estévez Valencia, P. (1997). Optimización mediante Algoritmos Genéticos. https://www.researchgate.net/publication/228708779

14. Flores López, R. y Fernández Fernández, J. M. (2008). Las redes neuronales artificiales ;fundamentos teóricos y aplicaciones prácticas (1ª ed.). Netbiblo.

15. Gómez Quesada, Fernández Graciani, M. Á., López Bonal, M. T. y Díaz-Marta, M. A. (1994). Aprendizaje con Redes Neuronales Artificiales. Ensayos: Revista de la Facultad de Educación de Albacete, 9, 169-180.

16. Haro Rivera, S., Zuñiga Lema, L., Meneses Freire, A., Vera Rojas, L. y Escudero Villa, A. (s. f.). Métodos de clasificación en Minería de Datos meteorológicos. 22/10/2018, 2(20), 7.

17. Hidalgo Sánchez, J. y Turrado Martínez, Jesús Ignacio. (s. f.). Algoritmos genéticos: Aplicación al problema de la mochila. 06/07/2021. https://www.it.uc3m.es/jvillena/irc/practicas/10-11/01mem.pdf

18. Hilera González, J. R. (s. f.). Nuevas técnicas de modelización y predicción de fenómenos complejos: Redes Neuronales Artificiales y Algoritmos Genéticos. https://dialnet.unirioja.es/servlet/articulo?codigo=570495

19. Hourcade, N., Coggan, V. y Della Mea, E. (2020). Sistema Informático de Patología Oncológica Músculo Esquelética [Tesis de grado, Universidad de la República, Facultad de Ingeniería]. https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/24357/1/HCD20.pdf

20. Izaurrieta, F. y Saavedra, C. (2000). Redes Neuronales Artificiales. Departamento de Física, Universidad de Concepción, Chile. https://disi.unal.edu.co/~lctorress/RedNeu/LiRna003.pdf

21. Kuri, A. (2000). Algoritmos Genéticos. Centro de Investigación en Computación. http://cursos.itam.mx/akuri/PUBLICA.CNS/2000/AGS.PDF

22. Larrañaga, P., Inza, I. y Moujahid, A. (s. f.). Redes Neuronales. http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf

23. Liñán Villafranca, L. y Merelo Guervós, J. J. (2021). DeepGProp [Tesis de grado, Universidad de Granada]. https://github.com/lulivi/deep-g-prop/archive/v1.0.0.zip. https://zenodo.org/record/4287505#.YQCDzxPHxLQ

24. Martínez Marhuenda, J. (2018). Iniciación al Entorno de Deep Learning Torch [Tesis de grado, Universitat Politècnica de València, Escola Tècnica Superior d'Enginyeria Informàtica]. https://m.riunet.upv.es/bitstream/handle/10251/109515/MART%C3%8DNEZ%20-%20Iniciaci%C3%B3n%20al%20entorno%20de%20deep%20learning%20Torch.pdf?sequence=1&isAllowed=y

25. Matich, D. J. (2001). Redes Neuronales: Conceptos básicos y aplicaciones. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

26. Melián, B., Moreno Pérez, J. A. y Moreno Vega, J. M. (2003). Metaheurísticas: Una visión global. Revista Iberoamericana de Inteligencia Artificial, 7(019), 1-28.

27. Miller, G. F., Todd, P. M. y Hedge, S. (1989). Designing Neural Networks using Genetic Algorithms. 7. https://doi.org/10.1016/0169-7439(93)80020-I

28. Modelo Vista Controlador (MVC). (s. f.). Universitat d'Alacant. Recuperado 7 de abril de 2021, de https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

29. Molino Minero Re, E., Cardoso Mohedano, J. G., Ruiz Fernández, A. C. y Sánchez Cabeza, J. A. (2014). Comparación de redes neuronales artificiales y análisis armónico para el pronóstico del nivel del mar (estero de Urías, Mazatlán, México). Ciencias Marinas, 40(4).

https://doi.org/10.7773/cm.v40i4.2463

30. Moreno Cantó, J. (2018). Arquitectura Domótica de bajo coste [Tesis de grado, Universidad de Alicante. Departamento de Ciencia de la Computación e Inteligencia Artificial]. http://rua.ua.es/dspace/handle/10045/81367

31. Moujahid, A., Inza, I. y Larrañaga, P. (s. f.). Algoritmos Genéticos. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco. http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t2geneticos.pdf

32. Nesmachnow, S. (2004). Algoritmos genéticos paralelos y su aplicación al diseño de redes de comunicaciones confiables [Tesis de maestría, Universidad de la República, Facultad de Ingeniería]. https://hdl.handle.net/20.500.12008/2932

33. Núñez Pölcher, P., Sala, H. y Matko, C. (2012). Herramientas de Software Libre para el análisis de datos científicos. Breve sinopsis y ejemplos aplicados al estudio del Sector Antártico Argentino. https://www.researchgate.net/publication/263619453

34. Pacheco, J. A. y Aragón, A. (2001). Análisis de algoritmos evolutivos para redes neuronales. Revista Electrónica de Comunicaciones y Trabajos, 2(1), 13.

35. Paz Jáuregui, C. M. (s. f.). Aplicación web didáctica para facilitar a niños de colegio a mejorar su fluidez en un idioma interactuado con niños de otros países [Tesis de grado, Universidad de Las Américas, Facultad de Ingeniería y Ciencias Agropecuarias]. https://www.researchgate.net/publication/263619453_Herramientas_de_Software_Libre_para_el_analisis_de_datos_cientificos

36. Pérez Lamancha, B. (2007). Estrategia de gestión de las pruebas funcionales en el Centro de Ensayo de Software. Revista Española de Innovación, Calidad e Ingeniería del Software, 3(3), 28-41. https://www.redalyc.org/pdf/922/92230304.pdf

37. Quintero Rodríguez, J. A. (2016). Desarrollo de una aplicación móvil multiplataforma de mensajería instantánea para agentes empresariales [Tesis de grado, Universidad Distrital Francisco José de Caldas, Facultad de Ingeniería]. https://bit.ly/3hk2kAN

38. Redes neuronales desde cero. (2019). [Blog]. IArtificial.net. https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/

39. Rivas Asanza, W. y Mazón Olivo, B. (2018). Redes neuronales artificiales aplicadas al reconocimiento de patrones (Editorial UTMACH). http://repositorio.utmachala.edu.ec/bitstream/48000/14223/1/Cap.1-Generalidades%20de%20las%20redes%20neuronales%20artificiales.pdf

40. Rivas Santos, V. M. (2003). Optimización de redes neuronales de funciones base radiales mediante algoritmos evolutivos [Tesis doctoral, Universidad de Granada]. https://dialnet.unirioja.es/servlet/tesis?codigo=12670

41. Rivas Cides, Á. (2019). Desarrollo y calibrado de un algoritmo evolutivo para la resolución de problemas de optimización [Tesis de grado, Universidad de Sevilla, Escuela Técnica Superior de Ingeniería]. http://bibing.us.es/proyectos/abreproy/92497/fichero/TFG-2497-RIVERO.pdf

42. Vivas, H., Pérez Mera, R. y Martínez Romero, H. J. (2014). Optimización en el entrenamiento del Perceptrón Multicapa [Universidad del Cauca, Facultad de Ciencias Naturales, Exactas y de la Educación]. http://matematicas.unicauca.edu.co/investigacion/gedi/optimizacion/TesisVivas.pdf

Capítulo X

Anexos

Anexos

"Es estéril y peligroso creer que uno domina el mundo entero gracias a internet cuando no se tiene la cultura suficiente que permite filtrar la información buena de la mala."

ZIGMUNT BAUMAN

Anexo I: Glosario de términos y tecnologías

Algoritmos genéticos
Un algoritmo genético es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Para ello, se parte de una población inicial de la cual se seleccionan los individuos más capacitados para luego reproducirlos y mutarlos para finalmente obtener la siguiente generación de individuos que estarán más adaptados que la anterior generación. Los pasos básicos de un algoritmo genético son: (1) Evaluar la puntuación de cada uno de los cromosomas generados. (2) Permitir la reproducción de los cromosomas siendo los más aptos los que tengan más probabilidad de reproducirse. (3) Con cierta probabilidad de mutación, mutar un gen del nuevo individuo generado. (4) Organizar la nueva población. (Arranz De la Peña & Parra Truyol, 2007).

Backpropagation
Es un algoritmo para hacer mejoras a los resultados en machine learning, donde la máquina aprende de sus errores y optimiza su funcionalidad de clasificación, adivinando otra solución al problema por sí mismo en caso de fallar.

Blueprints
Son módulos con los que se construyen las aplicaciones Flask que, aunque son similares entre sí, una aplicación puede tener un único objeto Flask, mientras que uno de estos últimos puede tener varios blueprints. Blueprint instrumenta el modelo MVC en una API y organiza los recursos para una mejor estructura.

Clasificadores
La clasificación, es la habilidad para adquirir una función que mapee (clasifique) un elemento de datos a una de entre varias clases predefinidas. Un objeto se describe a través de un conjunto de características (variables o atributos) x/{x1, x2 ,..., xn }. El objetivo de la tarea de clasificación es clasificar el objeto dentro de una de las categorías de la clase c = {c1 ,..., ck}. (Escudero villa et al., 2018).

DEAP
Esta librería es un framework de cálculo evolutivo para la creación rápida de prototipos. Busca la transparencia en la forma de su declaración y de codificación. Tiene implementado en su core el trabajo con tarjetas gráficas y trabaja en armonía con paralelizado o multiproceso.

Flask
Es un framework basado en Python con un motor de templates Jinja2 para la visualización. Su perfil minimalista acelera el proceso de implementación de un servidor web, además al estar implementando en python, permite poder utilizar las librerías de machine learning como Scikit o Pytorch, y devolver el resultado por medio de un JSON a cliente que lo solicita. La definición de webservices a nivel de Flask es cuestión de unas pocas líneas.

Gunicorn
Es un servidor web de python, de muy bajo consumo de recursos y sumamente rápido en términos de tiempo. Permite administrar peticiones simultáneas y a través de hooks, se pueden ejecutar

scripts de python en diferentes puntos de la aplicación.

Let's Encrypt
Es una autoridad de emisión de certificados gratuitos, automática y abierta, impulsada por la Linux Foundation, capaz de permitir el encriptado en las comunicaciones. Su utilización es intuitiva y capaz de concluirse en cuestión de unos minutos. Basta con indicarle el servidor sobre el que se reciben las peticiones y el nombre de dominio que se certifica.

Metaheurísticas
Son procedimientos de resolución de problemas de optimización en los que se tiene confianza de que su solución es de alta calidad. Estas soluciones heurísticas generadas a través de estos métodos, son factibles pero pueden no ser óptimas (Melián, 2003). La utilización de estos algoritmos en ocasiones estanca la solución en un óptimo local, pudiendo no alcanzar el óptimo global por tal motivo. Son algoritmos que nos aproximan a una solución, aunque para saber si es la óptima habría que medir esa diferencia.

Neurona artificial o Neuron
Una neurona artificial es un dispositivo que recibe una serie de entradas provenientes del exterior o de una capa anterior, produciendo una única salida al exterior o a todas las neuronas de la capa siguiente (Gómez Quesada et al., 1994).

Nginx
Es un servidor web, proxy inverso, caché de http y balanceador de carga, open-source, cuya principal característica es el rendimiento de un gran número de conexiones concurrentes. Además, ofrece un muy bajo uso de memoria, ya que no crea solicitudes en un único hilo sino que utiliza un enfoque asíncrono basado en eventos.

NumPy
Núñez Pölcher lo define como una colección de funciones y extensiones de Python, especializada en cálculo numérico y análisis de datos. Su potencial está en el tratamiento de grandes volúmenes de datos, lo que hace de esta librería una herramienta muy poderosa en machine learning al utilizar Python.

ONNX
Acrónimo del inglés de Open Neural Network Exchange, es un formato de representación de modelos de Machine Learning implementado originalmente por Facebook y Microsoft, al que luego fueron sumándose otras empresas importantes. El objetivo es poder intercambiar modelos de inteligencia artificial a través de un estándar, para que puedan ser consumidos por diferentes softwares independientemente de su lenguaje, similar a XML para la transferencia de información.

Pandas
Pandas es una herramienta de manejo de datos de alto nivel, construida con el paquete Numpy y cuya estructura de datos clave es el dataframe, que permite almacenar y manipular datos tabulados en filas de observaciones y columnas de variables. Para este proyecto, facilita el manejo de información de entrada en CSV de cara a pre procesar los datos y entrenar con un conjunto coherente y depurado.

Patrón MVC
El Modelo Vista Controlador o MVC, es una arquitectura de software que separa la aplicación en tres componentes distintos: la capa de presentación o Vista (V), la de acceso a datos o Modelo (M) y la intermediaria entre ambas, capa lógica o Controlador (C). Estamos ante un modelo muy utilizado que ha sabido adaptarse a todo tipo de aplicaciones, y sobre todo, a gran cantidad de lenguajes y plataformas de desarrollo. (1) El Modelo oficia de responsable de los datos que maneja el sistema, la lógica de acceso y sus mecanismos de persistencia. (2) La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos de interacción con éste. (3) El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno. (Servicio de Informática, s. f.).

Perceptrón multicapa
Para Vivas et al. (2014), un perceptrón multicapa es una red neuronal unidireccional constituida por tres o más capas, contando las de entrada y salida, es decir, que tiene al menos una capa oculta. Esta red aprende a través de minimización de una función de error, utilizando por ejemplo el algoritmo de back-propagation.

Pytorch
Es un framework de aprendizaje automático abierto basado en Python, que implementa y permite utilizar funciones de machine learning de manera sencilla e intuitiva. Además presenta dos principales ventajas, por un lado permite utilizar la GPU como unidad de cálculo de forma nativa, lo que acelera notablemente el proceso de entrenamiento de modelos y el tiempo de respuesta al cliente; como segundo punto a favor, posee una característica llamada paralelismo de datos, que le permite a esta librería distribuir el trabajo en los cores que tenga disponibles, tanto a nivel de CPU como de GPU.

Red neuronal Artificial
En el estudio de Romero (s. f.), se define una Red Neuronal Artificial (RNA) como un grafo dirigido, cuyos nodos se llaman elementos de proceso (EP), pueden tener cualquier número de conexiones y memoria local, y poseen una función de transferencia que, en función de las señales de entrada y la información de su memoria local, producen una señal de salida o altera su propia información local. Estos enlaces se llaman conexiones y funcionan como caminos unidireccionales instantáneos. Las redes neuronales tienen entradas que vienen del exterior, aunque sus salidas tienen información que sale desde la propia RNA.

Wheel
Es un formato de paquetes integrados de python cuyo objetivo es proporcionar la infraestructura estándar para gestionar los paquetes de distribución, logrando una interoperabilidad entre los distintos proyectos. Los archivos wheel son zip con extensión .whl. Una gran ventaja es que los paquetes van instalándose mientras se van desempaquetando, optimizando el tiempo que demora frente al desempaquetado e instalación tradicional.

Anexo II: Índice de figuras

Anexo III: Código de los operadores

Operador de reproducción Crossover

```
def operador_crossover(
   ind1: Gen,
   ind2: Gen
) -> Tuple[int, int]:
   # elige índices de capas aleatorias para intercambiar.
   layer_index = random.randint(0, len(ind1) - 1)

   cx_pts = random.sample(range(len(ind1.capas[layer_index].sesgos)), 2)

   (
       ind1.capas[layer_index].pesos[:, cx_pts[0]: cx_pts[1]],
       ind2.capas[layer_index].pesos[:, cx_pts[0]: cx_pts[1]],
   ) = (
       ind2.capas[layer_index].pesos[:, cx_pts[0]: cx_pts[1]].copy(),
       ind1.capas[layer_index].pesos[:, cx_pts[0]: cx_pts[1]].copy(),
   )
   (
       ind1.capas[layer_index].sesgos[cx_pts[0]: cx_pts[1]],
       ind2.capas[layer_index].sesgos[cx_pts[0]: cx_pts[1]],
   ) = (
       ind2.capas[layer_index].sesgos[cx_pts[0]: cx_pts[1]].copy(),
       ind1.capas[layer_index].sesgos[cx_pts[0]: cx_pts[1]].copy(),
   )
   return cx_pts, layer_index
```

Operador de mutación de capas

```
def mutador_capa(
    individual: Gen,
    rango_neuronas: Tuple[int, int]
) -> int:
    # Elegir aleatoriamente si agregar o quitar una capa, habiendo al menos dos en el modelo
    choice = 1 if len(individual) <= 2 else random.choice((-1, 1))

    difference = 0

    if choice > 0:
        # Elegir un número aleatorio de neuronas
        new_layer_output_neurons = random.randint(*rango_neuronas)
        # Obtener el número de neuronas de la última capa oculta
        previous_layer_output = individual.capas[-2].neuronas_salida
        # Agregar una última capa en el modelo
        individual.agregar_capa(
            Capa(
                nombre=f"Capa{len(individual)}",
                neuronas_entrada=previous_layer_output,
                neuronas_salida=new_layer_output_neurons,
            )
        )

        # Obtener las diferencias entre la nueva capa y la de salida para aplicar los cambios
necesarios para los cálculos
        output_layer_input_neurons = individual.capas[-1].pesos.shape[0]
        difference = new_layer_output_neurons - output_layer_input_neurons

        # Agregar los valores de las neuronas a la capa elegida
        if difference > 0:
            next_layer_neurons = len(individual.capas[-1].sesgos)
            individual.capas[-1].pesos = np.append(
                individual.capas[-1].pesos,
                np.random.uniform(-1.0, 1.0, (difference, next_layer_neurons)),
                axis=0,
            )
        # Eliminar los valores de las neuronas a la capa elegida
```

```
        elif difference < 0:
            individual.capas[-1].pesos = np.delete(
                individual.capas[-1].pesos,
                slice(
                    output_layer_input_neurons + difference,
                    output_layer_input_neurons,
                ),
                axis=0,
            )
        else:
        # Obtener los valores siguientes y eliminar la capa
        removed_predecessor_units = individual.capas[-3].neuronas_salida
        del individual.capas[-2]

        # Calcular la diferencia entre la capa a eliminary la de salida
        output_layer_input_len = individual.capas[-1].pesos.shape[0]
        difference = removed_predecessor_units - output_layer_input_len

        # Agregar las nauronas de entrada necesarias
        if difference > 0:
            next_layer_neurons = len(individual.capas[-1].sesgos)
            individual.capas[-1].pesos = np.append(
                individual.capas[-1].pesos,
                np.random.uniform(-0.5, 0.5, (difference, next_layer_neurons)),
                axis=0,
            )
        # Eliminar el excedente
        elif difference < 0:
            individual.capas[-1].pesos = np.delete(
                individual.capas[-1].pesos,
                slice(
                    output_layer_input_len + difference, output_layer_input_len
                ),
                axis=0,
            )

    # Actualizar las neuronas de entrada de la capa de salida
    individual.capas[-1].neuronas_entrada += difference

    return choice
```

Operador de mutación de neuronas

```
def mutador_neurona(
    individual: Gen
) -> int:
    # Las capas de entrada y salida son fijas, por lo que se agregan o quitan son las capas
ocultas
    layer_index = random.randint(0, len(individual) - 2)

    # Elige una capa al azar para añadir una neurona más
    choice = 1 if len(
        individual.capas[layer_index].sesgos) <= 2 else random.choice((-1, 1))

    if choice > 0:
        # Toma la información de la capa anterior para crear el nuevo individuo
        previous_layer_neurons = individual.capas[layer_index].pesos.shape[0]

        # Agregar la neuva neurona a los pesos y sesogs de la capa seleccionada
        individual.capas[layer_index].pesos = np.append(
            individual.capas[layer_index].pesos,
            np.random.uniform(-0.5, 0.5, (previous_layer_neurons, 1)),
            axis=1,
        )
        individual.capas[layer_index].sesgos = np.append(
            individual.capas[layer_index].sesgos,
            [random.uniform(-0.5, 0.5)],
            axis=0,
        )
        # Añadir la nueva neurona a la capa escogida
```

```
            next_layer_neurons = len(individual.capas[layer_index + 1].sesgos)
            individual.capas[layer_index + 1].pesos = np.append(
                individual.capas[layer_index + 1].pesos,
                np.random.uniform(-0.5, 0.5, (1, next_layer_neurons)),
                axis=0,
            )
        else:
            # Eliminar la última neurona de los pesos y los sesgos
            individual.capas[layer_index].pesos = np.delete(
                individual.capas[layer_index].pesos, -1, axis=1
            )
            individual.capas[layer_index].sesgos = np.delete(
                individual.capas[layer_index].sesgos, -1, axis=0
            )
            # Eliminar la última neurona de la próxima capa
            individual.capas[layer_index + 1].pesos = np.delete(
                individual.capas[layer_index + 1].pesos, -1, axis=0
            )

        # Actualizar el valor de la capa actual y la próxima
        individual.capas[layer_index].neuronas_salida += choice
        individual.capas[layer_index + 1].neuronas_entrada += choice

        return choice
```

Operador de mutación de pesos y sesgos

```
def mutador_pesos_sesgos(
    individual: Gen,
    attribute: str,  # define qué es lo que se va a mutar
    gen_prob: float
) -> int:
    ''' Operador de mutación de pesos y sesgos '''
    mutated_genes = 0

    for capa in individual.capas:
        weights = getattr(capa, attribute)
        weights_shape = weights.shape
        mask = np.random.rand(*weights_shape) < gen_prob
        mutated_genes += np.count_nonzero(mask)
        mutations = np.random.uniform(-0.5, 0.5, weights_shape)
        mutations[~mask] = 0
        weights += mutations

    return mutated_genes
```

Leroy Deniz

API para entrenamiento de redes neuronales profundas evolucionadas a través de algoritmos genéticos

Trabajo Final de Grado
Universidad del País Vasco
Septiembre 2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA