UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

ExoplanetIA
Machine Learning para la detección de exoplanetas
Documentación Técnica

Presentado por Jesús María Herruzo Luque
en Universidad de Burgos — 24 de junio de 2020
Tutor: Carlos López Nozal

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este punto pasamos a detallar la planificación del proyecto y los pasos que se han dado para poder completarlo con éxito.

A.2. Planificación temporal

La planificación y el desarrollo de este proyecto se han llevado a cabo siguiendo una metodología ágil, concretamente scrum, teniendo que en cuenta que la plataforma de desarrollo usada, GitLab, presenta una nomenclatura propia que no siempre coincide con la de scrum. Así, por ejemplo, GitLab usa *milestones* o *issues* para referirse a *sprints* o *product backlog items (PBIs)*. En este documento intentaremos usar los terminos de scrum siempre que sea posible.

En relación a la plataforma, otro punto a reseñar es la no disponibilidad de gráficas de *burndown* en la versión gratuita. Se ha intentado usar otras alternativas, como Screenful, pero requiere acceso completo a la API de GitLab. Estando el proyecto alojado en la cuenta privada de la empresa HP SCDS, esta no es una opción viable. Así pues, la realización del proyecto se ha llevado a cabo sin disponer de estos gráficos.

El proyecto se ha llevado a cabo siguiendo un sprint de dos semanas, al final de las cuales, en reunión con los tutores, tanto por parte de la Universidad de Burgos como por parte de HP SCDS, se mostraba el trabajo realizado, se proponian cambios o mejoras, y se establecian los objetivos

para el siguiente sprint. El seguimiento de los PBIs se ha realizado a través del tablero Kanban que GitLab provee por defecto.

Pasamos a ver con más detalle el trabajo realizado en cada uno de los sprints:

01 - Arranque del proyecto [10/02/2020 -– 17/02/2020]

El sprint de arranque del proyecto es fundamental teórico, dedicado a estudiar y decidir como vamos a hacer el proyecto, con que herramientas y que medidas usaremos para determinar la validez de un modelo. Este sprint inicial solamente duró una semana, para de esta forma, trás la primera reunión con los tutores el 17 de febrero, poder comenzar un nuevo sprint.

- Estudio comparativo de librerías de machine learning: a la hora de abordar la realización del proyecto nos encontramos con multitud de librerías disponibles. Como primera tarea seleccionamos varias de ellas como posibles candidatas y procedemos a estudiar sus características distintivas.

- Análisis estadístico: en esta tarea nos dedicamos a estudiar métricas alternativas a la precisión para valorar nuestro modelo, como la sensibilidad y la especificidad, así las curvas ROC *receiver oOperating characteristic, o característica operativa del receptor*.

- Estudio documentación de un proceso experimental con CRISP_DM: aquí aprendemos una metodología habitúal a la hora de trabajar en ciencias de datos que usaremos durante el proyecto.

- Elección de librerias: una vez disponemos de los datos necesarios, valoramos como cada librería se adapta a las necesidades del proyecto y seleccionamos una, Pytorch.

- Estudio teórico del perceptrón multicapa: como última tarea del sprint, estudiamos el modelo básico de red neuronal profunda, el perceptrón multicapa.

02 - Desarrollo del perceptrón [02/03/2020 – 16/03/2020]

En este sprint nos enfocamos en la parte formal del TFG, estudiando LaTex y las plantillas de la memoria así como la estructura de archivos, y en los primeros pasos con Pytorch y el dataset.

- Estudio de LaTex y plantillas de memoria: estudiamos los documentos a entregar y sus diferentes secciones a la vez que preparamos el entorno para trabajar con documentos LaTex.
- Añadir a la memoria LaTex la documentación generada en makdown: pasamos la documentación generada en el primer sprint, escrita en markdown en la wiki del proyecto, a LaTex y la incorporamos a la memoria.
- Estructurar las carpetas del repositorio y actualizar fichero Readme.md: procedemos a estructurar los archivos de nuestro solución siguiendo ejemplos de proyectos de exito y otros TFG.
- Aprendizaje de Pytorch: en este punto nos centramos en aprender la librería que vamos a usar.
- Procesado de los datos: finalmente estudiamos los datos con los que vamos a trabajar y realizamos una implementación básica de red.

03 - Modelo de perceptrón [16/03/2020 - 06/04/2020]

Por circunstancias personales imprevistas, este sprint se alarga una semana más. En este tiempo apenas se puede avanzar en el proyecto; solo se completa una tarea y se comienza a trabajar en el modelo básico del perceptrón aunque no se completa.

- Gestión de la configuración del repositorio GitLab, donde limpiamos el repositorio de ficheros temporales de Python y Pytorch

04 - Finalizar perceptrón [07/04/2020 - 20/04/2020]

En este sprint nos centramos en el desarrollo de los diversos modelos del perceptrón.

- Modelo simple de perceptron: terminamos la tarea comenzada en el sprint anterior donde generamos nuestro modelo base de percetrón probando con los algoritmos SGD y Adam.
- Entrenar perceptrón con BCEWithLogitsLoss: ampliamos nuestro modelo inicial para usar la función de perdida BCEWithLogitsLoss.
- Mejorar modelo de perceptrón: terminado nuestro modelo base, pasamos ahora a incluir el filtro gaussiano.
- Añadir SMOTE en el modelo de perceptrón: procedemos a entrenar nuevos modelos de percetrón usando SMOTE.
- Hacer análisis de frecuencia con la transformada de Fourier: para terminar con los modelos basados en perceptrón, cambiamos el análisis de la intensidad de luz por el de la frecuencia.
- Caso de estudio de documentación TFG en LaTex relacionado: comprobamos otros TFG para estudiar diferentes enfoques a la hora de encarar la escritura de la memoria.
- Crear notebook para testear los modelos: para finalizar el sprint, creamos un notebook donde poder ejecutar nuestros modelos contra el dataset de testing.

05 - Probar LSTM y crear web [21/04/2020 - 04/05/2020]

En este sprint nos marcamos como objetivo desarrollar modelos basados en redes LSTM y crear una primera versión de aplicación web para poder ejecutar los modelos.

- Formación en conjuntos de datos desequilibrados: estudiamos algunos métodos adicionales para trabajar con dataset desbalanceados.
- Integración documentación Wiki a memoria LateX: migramos el resto de información que teniamos en la wiki a la memoria.
- Crear modelo de red LSTM: desarrollmos varios modelos usando redes LSTMs.
- Crear web para testear modelos: creamos una primera versión donde sea posible subir un archivo para probar la ejecución del modelo.

- Escribir capítulos de Introducción y Objetivos de la memoria: pasamos a completar los capítulos iniciales de la memoria.

06 - Preparar memoria [05/05/2020 - 18/05/2020]

Estaba previsto que el foco de este sprint fuese trabajar en la memoria pero algunos problemas en las tareas relacionadas con la aplicación web provocan que no haya mucho avance en la documentación.

- Desplegar web en Heroku: desplegamos de forma manual la web en Heroku para comprobar que todo funciona correctamente.
- Crear pipeline de release: después del despliegue manual procedemos a automatizar el proceso generando un pipeline en GitLab.
- Completar web: añadimos algunas páginas de información sobre la misión Kepler, sobre los datos y sobre el proyecto. También la localizamos soportando el idioma inglés.
- Ampliar descripcion de las herramientas usadas: añadimos más detalles sobre las herramientas usadas en el proyecto.

07 - Anexos [19/05/2020 – 31/05/2020]

Entrando ya en las etapas finales, este sprint se dedica a continuar el trabajo en la memoria y a ir puliendo detalles del proyecto en general.

- Documentar proceso experimental: completamos el apartado quinta de la memoria detallando las lecciones aprendidas durante el proyecto.
- Incluir dataset y datos del modelo en la web: ampliamos la web incluyendo un dataset de pruebas, información sobre el modelo desplegado y enlazando al repositorio original de los datos en Kaggle.
- Evaluar la calidad del código: usamos una herramienta para controlar la calidad de nuestro código, Codebeat.
- Añadir badges al repositorio: incluimos algunos badges en el repositorio y en el Readme.md para conocer la calidad del código, como fue la ejecución del pipeline de CD o como se encuentra la web en Heroku.

A.3. Estudio de viabilidad

Abordamos a continuación un breve estudio de la viabilidad tanto legal como económica de este proyecto.

Viabilidad económica

Este proyecto nunca se ha enfocado a una viabilidad económica y sería difícil venderlo *tal cual*. Quizá la razón más importante es que apenas hay demanda; el conocimiento de que una estrella es orbitada o no por exoplanetas es, a día de hoy, de poca utilidad comercial, por lo que hay pocas, por no decir ninguna, empresas dispuestas a pagar por ello.

A parte de las empresas privadas, tenemos a diferentes organismos públicos interesados en la exploración espacial, como la NASA, la ESA, la CNSA o similares. Estas agencias se enfrentan a un grave problema común: hay muchos estrellas con posibles exoplanetas y disponen de pocos recursos para explorarlos todos. Dado el carácter probabilístico de este tipo de modelos, puede tener sentido comercial disponer de diferentes aplicaciones y enfocar sus recursos en estudiar aquellas estrellas seleccionadas como más probables por todos, o la mayoría, de las aplicaciones de las que dispongan. A nivel de modelos, en aprendizaje automático, tenemos los metodos de *ensemble* que, básicamente, consisten en combinar varios modelos base para producir otro modelo óptimo. Así pues, no sería ilógico realizar un proceso similar a nivel de aplicaciones.

Por otro lado, otro colectivo que podría estar interesado en nuestra aplicación sería la comunidad científica, más alla de aquellos que trabajan en las agencias espaciales. La confirmación de la presencia o no de exoplanetas en torno a diferentes estrellas puede ayudar a generar nuevo conocimiento sobre este tipo de sistemas, sirviendo de base para confirmar hipótesis o para descartar teorías erróneas.

Finalmente, tenemos otro colectivo que podría estar interesado en nuestro proyecto, los aficionados a la astronomía. En este caso, se podría complementar la aplicación con alguna funcionalidad más como, por ejemplo, proporcionar las coordenadas de la estrella en la que se han identificado exoplanetas de forma que el aficionado pueda apuntar su telescopio diréctamente a ella.

Viabilidad legal

En lo referente a la viabilidad legal del proyecto debemos separarla en dos puntos: las herramientas y los datos. Respecto a las primeras, no

se encuentra ningún problema. Todas las herramientas pueden usarse en aplicaciones comerciales, presentando licencias permisivas (BSD o MIT en su mayoría).

En el apartado de los datos si podemos encontrarnos con mayor problema. Los usados para el entrenamiento de estos modelos, pertenecientes a la campaña 3, son de dominio público, ofrecidos por el Mikulski Archive [5], esperando sólamente ser citados por investigadores. Sin embargo, si deseamos mejorar nuestros modelos con datos más recientes de otras campañas si podemos encontrarnos con datos restringidos o con copyright de forma temporal. Además, en el caso de querer implementar la versión para aficionados a la astronomía, el catálogo de estrellas si tiene copyright.

Apéndice B

Especificación de Requisitos

B.1. Introducción

A continuación, procedemos a abordar los requisitos del proyecto.

B.2. Objetivos generales

Este proyecto de investigación nace con los siguientes objetivos:

- Investigar las técnicas y soluciones existentes de aprendizaje automático para la detección de exoplanetas mediante el análisis del flujo de luz.
- Estudiar los datos disponibles, las diferentes formas de procesarlos y como este procesado puede ayudarnos a obtener mejores resultados.
- Diseñar, implementar y comparar diferentes soluciones, seleccionando el modelo que presente mejores resultados.
- Disponer de software que facilite el uso de diferentes técnicas y el desarrollo de modelos alternativos a los aquí presentados.

Además, durante el desarrollo del proyecto se considera añadir algunos nuevos requisitos de importancia secundaria:

- La creación de una interfaz gráfica que permita ejecutar el modelo para comprobar los datos contenidos en un archivo.

- Desarrollar un pipeline de despliegue continuo que permita desplegar de forma automatica cualquier cambio en nuestro modelo o en la interfaz grafica.

B.3. Catalogo de requisitos

Se muestran a continuación las características que debe cumplir nuestro proyecto, clasificándolas en dos grupos, requisitos funcionales y no funcionales. Mientras que los primeros definen un comportamiento exacto del software, es decir, *que* hace, los segundos hacen referencia a sus propiedades, es decir, *como* las hace.

Definimos dos actores:

- **Investigador**: persona que realiza la investigación y utiliza el software para generar y entrenar modelos.
- **Usuario**: persona que usa el software para determinar si un conjunto de estrellas tienen o no exoplanetas orbitando en torno a ellas.

Requisitos funcionales

- **RF-1** El investigador debe poder dividir un conjunto de datos en sets de entrenamiento y validación.
  - **RF-1.1** Se debe indicar que porcentaje respecto al total de instancias se incluiran en el set de validación.
  - **RF-1.2** La inclusión de una instancia concreta en un set u otro debe ser aleatoria pero se podrá indicar una semilla para que, dado un set de entrada concreto y un porcentaje determinado, la división produzca los mismos set de entrenamiento y validación.
- **RF-2** El investigador debe poder aplicar técnicas y algoritmos que prepararen los datos.
  - **RF-2.1** Debe poder normalizar los datos.
  - **RF-2.2** Debe poder aplicar un filtro gaussiano.
  - **RF-2.3** Debe poder eliminar los picos anormales de intensidad de luz.

    - **RF-2.4** Debe poder transformar el conjunto de datos desde el dominio de la intensidad al dominio de la frecuencia.

- **RF-3** El investigador debe poder mostrar gráficas de la frecuencia o intensidad de la luz respecto al tiempo.

- **RF-4** El investigador debe poder instanciar modelos de redes neuronales.

    - **RF-4.1** Debe poder instanciar modelos de perceptrón multicapa configurables.
    - **RF-4.2** Debe poder instanciar modelos de red LSTM configurable.

- **RF-5** El investigador debe poder entrenar un modelo.

    - **RF-5.1** El investigador debe poder elegir diferentes algoritmos de optimización o funciones de coste.
    - **RF-5.2** Debe mostrarse la evolución del entrenamiento.
    - **RF-5.3** Debe guardarse el mejor modelo generado durante el entrenamiento según el nombre y la ruta indicada por el investigador.
    - **RF-5.4** El investigador debe poder mostrar gráficamente los resultados del entrenamiento.

- **RF-6** El investigador debe poder probar un modelo guardado.

    - **RF-6.1** El investigador debe poder indicar el nombre y la ruta del modelo a cargar.

- **RF-7** El usuario debe poder analizar un archivo con datos en la aplicación.

    - **RF-7.1** El usuario debe tener información suficiente sobre el formato del archivo a procesar.
    - **RF-7.2** El usuario debe obtener que estrellas presentan exoplanetas.
    - **RF-7.3** La aplicación debe mostrar las gráficas correspondientes a las estrellas con exoplanetas.

Requisitos no funcionales

- **RNF-1** El entrenamiento o el testeo de modelos debe poder llevarse a cabo tanto en CPUs como en GPUs, según el hardware del que el investigador disponga.
- **RNF-2** Debe proveerse una aplicación para que un usuario sin entorno ni conocimientos de programación puede probar alguno de los modelos entrenados.

B.4. Especificación de requisitos

Diagrama de casos de uso del investigador

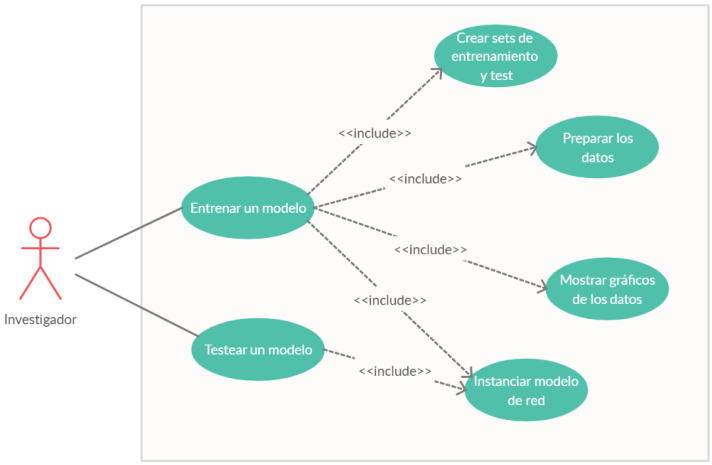

Figura B.1: Diagrama de casos de uso del investigador

Diagrama de casos de uso del usuario

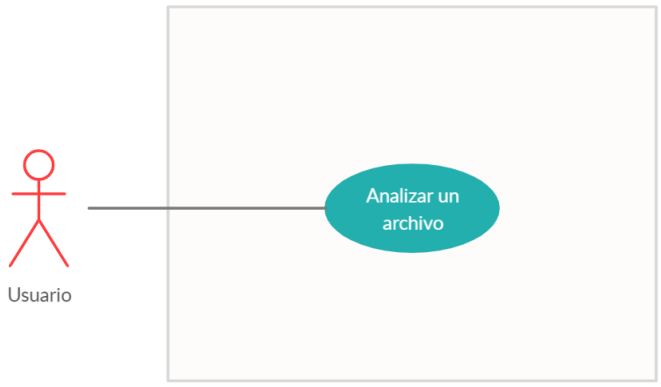

Figura B.2: Diagrama de casos de uso del usuario

Especificaciones de los casos de uso

Caso de uso	Entrenar un modelo
Requisitos	RF-5 RF-5.1 RF-5.2 RF-5.3 RF-5.4
Descripción	El investigador puede entrenar un nuevo modelo dado un conjunto de datos usando un algoritmo de optimización y una función de coste.
Precondiciones	El investigador debe disponer de un conjunto de datos.
Acciones	1. El investigador carga los datos. 2. Opcionalmente, el investigador puede dividir sus datos en dos conjuntos distintos, entrenamiento y validación. 3. Opcionalmente, el investigador puede preparar los datos usando alguna de las técnicas y herramientas propuestas. 4. Opcionalmente, el investigador visualizar los datos. 5. El investigador elige un tipo de modelo y genera una instancia. 6. El investigador fija los hiperparámetros según deseé. 7. El investigador decide que algoritmo de optimización y función de coste usar. 8. El investigador comienza el entrenamiento. 9. Opcionalmente, cuando el entrenamiento termine, el investigador puede mostrar gráficamente los resultados del mismo.
Postcondiciones	Se ha generado un modelo y las imagenes con los resultados del entrenamiento.
Excepciones	Formato de datos incorrecto. Algoritmo de optimización invalido. Función de coste invalida.Configuración de la instancia del modelo incorrecta.
Importancia	Alta

Tabla B.1: Caso de uso ”Entrenar un modelo”.

Caso de uso	Crear sets de entrenamiento y validación
Requisitos	RF-1 RF-1.1 RF-1.2
Descripción	A partir de un conjunto de datos, el investigador puede dividirlo en dos de forma aleatoria y segun la proporción deseada.
Precondiciones	El investigador debe disponer de un conjunto de datos.
Acciones	1. El investigador selecciona el conjunto de datos. 2. El investigador elige que porcentaje del total ocupará el set de validación 3. Opcionalmente, el investigador elige una semilla para la selección aleatoria.
Postcondiciones	El investigador obtiene los datos divididos en dos conjuntos.
Excepciones	El conjunto de datos está vacio. La proporción es uno o mayor.
Importancia	Media.

Tabla B.2: Caso de uso "Crear sets de entrenamiento y validación".

Caso de uso	Preparar los datos
Requisitos	RF-2 RF-2.1 RF-2.2 RF-2.3 RF-2.4
Descripción	El investigador debe poder aplicar a los datos las funciones usadas durante el entrenamiento: normalización, eliminación de picos de intensidad, filtro gaussiano y transformada de Fourier.
Precondiciones	El investigador debe disponer de un conjunto de datos.
Acciones	1. El investigador selecciona el conjunto de datos. 2. El investigador aplica la función correspondiente.
Postcondiciones	El conjunto de datos transformado.
Excepciones	El conjunto de datos está vacio.
Importancia	Media.

Tabla B.3: Caso de uso "Preparar los datos".

Caso de uso	Mostrar gráficos de los datos
Requisitos	RF-3
Descripción	El investigador puede mostrar la gráfica de cualquiera de las estrellas incluidas en el conjunto de datos.
Precondiciones	El investigador debe disponer de un conjunto de datos.
Acciones	1. El investigador selecciona el conjunto de datos. 2. El investigador indica los indices de las estrellas a mostrar. 3. El investigador aplica la función.
Postcondiciones	Gráfica(s) mostrando los datos correspondientes.
Excepciones	El indice de las estrellas seleccionadas no se encuentra en el conjunto de datos.
Importancia	Baja.

Tabla B.4: Caso de uso "Mostrar gráficos de los datos".

Caso de uso	Instanciar modelo de red
Requisitos	RF-4 RF-4.1 RF-4.2
Descripción	El investigador debe poder instanciar modelos de perceptrón o de LSTM.
Precondiciones	Ninguna.
Acciones	1. El investigador selecciona el tipo de modelo a instanciar. 2. Opcionalmente, el investigador los parámetros para el modelo.
Postcondiciones	Instancia del modelo.
Excepciones	Ninguna.
Importancia	Alta.

Tabla B.5: Caso de uso "Instanciar modelo de red".

Caso de uso	Testear un modelo
Requisitos	RF-6 RF-6.1
Descripción	El investigador debe poder hacer predicciones sobre un conjunto de datos con un modelo previamente entrenado.
Precondiciones	El investigador debe contar un modelo entrenado guardado en disco.
Acciones	1. El investigador selecciona el conjunto de datos. 2. El investigador instancia un modelo de red acorde al modelo que desee testear. 3. El investigador testea el modelo.
Postcondiciones	Matriz de confusión del modelo.
Excepciones	El modelo de red instanciado no es el adecuado para el modelo guardado.
Importancia	Alta.

Tabla B.6: Caso de uso "Testear un modelo".

Caso de uso	Analizar un archivo
Requisitos	RF-7 RF-7.1 RF-7.2 RF-7.3
Descripción	El usuario debe poder cargar un archivo en la aplicación, obteniendo predicciones sobre que estrellas presentan exoplanetas así como las gráficas de dichas estrellas.
Precondiciones	El usuario debe disponer de un archivo con datos.
Acciones	1. El usuario selecciona el archivo de datos. 2. El usuario carga el archivo en la aplicación.
Postcondiciones	Resultado del análisis con las gráficas correspondientes.
Excepciones	El archivo no es válido.
Importancia	Media.

Tabla B.7: Caso de uso "Analizar un archivo".

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado abordamos el diseño del software generado durante el proyecto, tanto la pate dedicada al proyecto investigador como la aplicación web.

C.2. Diseño de datos

A la hora de estudiar el diseño de nuestro proyecto, los datos juegan un papel fundamental al ser la materia prima que nos permitirá elaborar nuestros modelos. Estos se pueden encontrar en la plataforma *Kaggle* para su consulta y descarga. Examinándolos constatamos que viene definidos en dos archivos de texto plano, *exoTrain.csv* y *exoTest.csv*, con valores separados por comas. Detallamos a continuación su formato:

- *exoTrain.csv*:

- 5087 filas u observaciones
- 3198 columnas o características
- La primera columna es la etiqueta para clasificación. Las columnas 2-3198 son los valores de flujo a lo largo del tiempo
- Hay 37 estrellas con exoplanetas confirmados y 5050 estrellas sin exoplanetas

- *exoTest.csv*:

  - 570 filas u observaciones
  - 3198 columnas o características
  - La primera columna es la etiqueta para clasificación. Las columnas 2-3198 son los valores de flujo a lo largo del tiempo
  - Hay 5 estrellas con exoplanetas confirmados y 565 estrellas sin exoplanetas

Este formato de archivo determina parte de las estructura de nuestros modelos ya que, salvo que optemos por eliminar características, la capa de entrada debe contar con el mismo numero de entradas que columnas de datos, esto es, 3197. A su vez, la aplicación web desplegada debe aceptar archivos que sigan estan formato para poder analizarlos correctamente.

Los valores correspondientes a la intensidad de luz presentan una amplia variedad, moviendose generalmente entre los intervalos de -10^5 a 10^5.

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8
count	5087.000000	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03	5.087000e+03
mean	1.007273	1.445054e+02	1.285778e+02	1.471348e+02	1.561512e+02	1.561477e+02	1.469646e+02	1.168380e+02	1.144983e+02
std	0.084982	2.150669e+04	2.179717e+04	2.191309e+04	2.223366e+04	2.308448e+04	2.410567e+04	2.414109e+04	2.290691e+04
min	1.000000	-2.278563e+05	-3.154408e+05	-2.840018e+05	-2.340069e+05	-4.231956e+05	-5.975521e+05	-6.724046e+05	-5.790136e+05
25%	1.000000	-4.234000e+01	-3.952000e+01	-3.850500e+01	-3.505000e+01	-3.195500e+01	-3.338000e+01	-2.813000e+01	-2.784000e+01
50%	1.000000	-7.100000e-01	-8.900000e-01	-7.400000e-01	-4.000000e-01	-6.100000e-01	-1.030000e+00	-8.700000e-01	-6.600000e-01
75%	1.000000	4.825500e+01	4.428500e+01	4.232500e+01	3.976500e+01	3.975000e+01	3.514000e+01	3.406000e+01	3.170000e+01
max	2.000000	1.439240e+06	1.453319e+06	1.468429e+06	1.495750e+06	1.510937e+06	1.508152e+06	1.465743e+06	1.416827e+06

Figura C.1: Algunos datos estadísticos sobre el dataset de entrenamiento

Por otro lado, la parde de investigación de nuestro proyecto no utiliza gestores de bases de datos. La salida de la información procesada se corresponde con modelos entrenados y con las imagenes de los resultados de dichos entrenamientos. Todos ellos son archivos que se almacenan en disco en función del nombre que el investigador asigne al modelo.

De forma similar, la aplicación web no interacciona con ningún gestor de bases de datos y ni siquiera almacena datos. Los resultados del análisis y las gráficos mostrados son generados de forma dinámica según sean necesarios y presentados al cliente trás lo cual son descartados.

C.3. Diseño procedimental

A la hora de realizar un nuevo experimento, el investigador dispone de bastante libertad a la hora de elegir diversas técnicas y algoritmos. Sin embargo, el flujo general de trabajo se puede describir de forma general para englobar la mayoría de las casuísticas. Como tal, los diagramas aquí presentandos reflejan una guía orientativa con la que se han desarrollado los experimentos de este proyecto y con los que otro investigador podría desarrollar experimentos nuevos.

La creación de un experimento es un proceso bastante linear: cargar los datos, aplicar las técnicas adecuadas para preparar los datos, configurar el modelo y realizar el entrenamiento. El proceso puede verse en la imagen Diagrama de flujo de la realización de un experimento C.2.

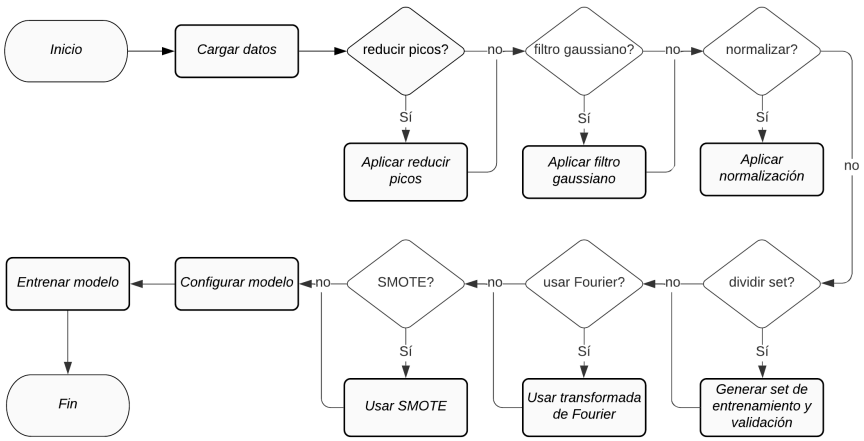

Figura C.2: Diagrama de flujo de la realización de un experimento

Por su mayor complejidad y su importancia en el proyecto, nos interesa estudiar de forma más detallada el proceso de entrenamiento del modelo, proceso que puede apreciarse en la imagen Diagrama de flujo del proceso de entrenamiento C.3.

El proceso arranca creando las instancias de las clases necesarias, una del tipo *resultado* y otra de un tipo específico de Pytorch, *dataloader*, que suministra los *batches* necesarios según orqueste nuestro *fluxdataset*. En caso de suministrar un set de validación a la función, se prepara para ser procesado.

A continuación se itera hasta que se alcancen todos los epochs establecidos. En cada epoch, el *dataloader* irá suministrando *batches* conteniendo los datos a procesar. En este proyecto se ha fijado el tamaño del *batch* a uno, por lo que vamos iterando los datos una estrella cada vez. En este bucle de segundo nivel es donde se realiza el entrenamiento propiamente dicho. Primero, el modelo hace una predicción sobre los datos de la estrella correspondiente. A continuación, se comprueba la perdida de dicha predicción para proceder a actualizar los pesos del modelo, intentando mejorar la proxima predicción.

Cuando se han completado todos los *batches*, nuestro objeto resultados almacena la información de esta ejecución y pasamos a comprobar nuevamente si tenemos set de validación. Si no es el caso, comprobariamos si cumplidos los epochs necesarios o debemos seguir entrenado. En el caso de tener set de validación, tenemos que comprobar el desempeño del modelo. Para ello, nuestro modelo realiza una predicción sobre todo el conjunto de validación, obtiene la perdida correspondiente y calcula las puntuaciones del modelo, como la sensibilidad y la especificidad. Si esta puntuación es mejor que la que teniamos, guardamos el nuevo modelo. Posteriormente, en cualquiera de los dos casos, volvemos a comprobar si hemos llegado al final de los epochs o debemos seguir entrenando.

Cuando llegamos al final de los epochs, el proceso muestra los datos del entrenamiento y finaliza.

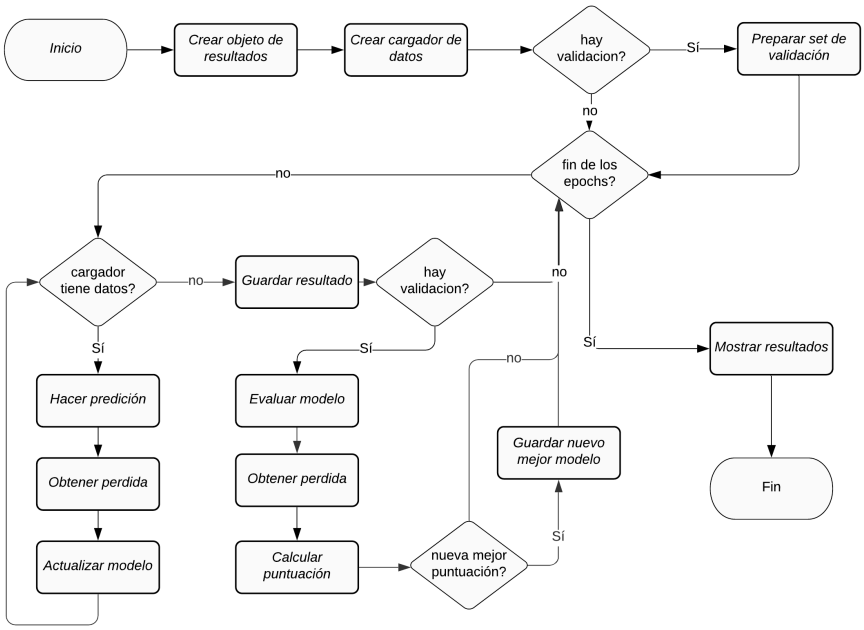

Figura C.3: Diagrama de flujo del proceso de entrenamiento

C.4. Diseño arquitectónico

La arquitéctura de nuestro proyecto es relativamente simple. Dado el foco en la investigación y en probar nuevos métodos y técnicas conforme el proceso avanza, se prefiere mantener el código más flexible posible para evitar tener que deshacerlo a posteriori si el proyecto continua por una linea diferente.

A nivel de entidades, nuestro proyecto define y utiliza las siguientes:

- **modelo_perceptron**: define la arquitectura base de un modelo de perceptrón multicapa, permitiendo configurar el número de neuronas de sus capas internas y de salida.
- **modelo_lstm**: define la arquitectura base de un modelo de red LSTM, permitiendo configurar el tamaño de sus capas, el numero de capas internas y determinar un porcentaje de *dropout*.
- **fluxdataset**: implementación de la clase abstracta *dataset* de Pytorch que proporciona funcionalidad para gestionar la carga de datos durante un entrenamiento.

- **resultados**: gestiona los resultados que se generan durante un entrenamiento, permitiendo imprimirlos en formato gráfico.
- **utils**: módulo genérico con funciones para procesar los datos y entrenar un modelo.

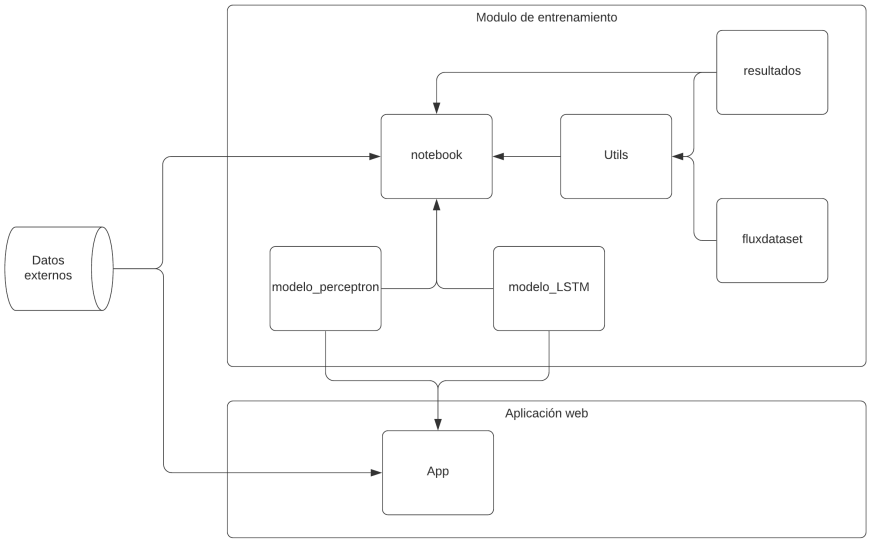

Figura C.4: Diagrama de clases de la aplicación

Además, se incluyen los siguientes notebook (denominados con el nombre genérico notebook en la imagen) donde se han llevado a cabo los experimentos:

- **perceptron_base**: experimentos realizados con perceptrones multicapa con datos en brutos, con reducción de picos de flujo, normalizados y con filtros de Gauss.
- **perceptron_smote**: experimentos con perceptrones multicapa usando SMOTE.
- **perceptron_fourier**: experimentos con perceptrones multicapa realizando análisis de frecuencias.
- **lstm_base**: experimentos con redes LSTM usando procesado de datos y SMOTE.

- **comparador**: este notebook no contiene experimentos, sino que permite cargar un modelo entrenado para testearlo. También se encuentra incrustado un documento html donde se recogen las puntuaciones de los modelos durante el entrenamiento.

Por último comentaremos brevemente la estructura de directorios y los ficheros que componen la aplicación web:

- **models**: directorio que contiene los modelos que se pueden probar en la web.
- **static**: directorio con el contenido estático de la web: imagenes, hojas de estilos y el dataset de pruebas.
- **templates**: documentos html que componen la web, siendo *base.html* la plantilla por defecto para todos ellos.
- **translations**: directorio con los archivos para la localizacion.
- **.flaskenv**: archivo de configuración de flask
- **app.py**: archivo principal de la aplicación, donde se definen las routas accesibles por los clientes así como las acciones a realizar en cada una ellas. Dado el reducido tamaño de la web, no se han separado en archivos y directorios diferentes para seguir un patrón MVC.
- **config.py**: archivo de configuración de la web.
- **forms.py**: definición del formulario para subir el archivo.
- **modelo_perceptron**: definición del perceptrón multicapa para poder cargar modelos basados en él.
- **modelo_lstm**: definición de red LSTM para poder cargar modelos basados en ella.

Apéndice D

Documentación técnica de programación

D.1. Introducción

Comentaremos aquí los detalles más técnicos sobre el proyecto de forma que sea posible para cualquiera recrear los experimentos y continuar el trabajo aquí presentado.

D.2. Estructura de directorios

El código fuente del proyecto está organizado en la siguiente estructura de directorios:

- **/**: directorio raiz del proyecto donde encontramos los ficheros de configuración para git, docker, la aplicación web, así como los archivos necesarios para recrear el entorno de trabajo y la descripción del proyecto.
- **/app/**: aplicación web con los ficheros de inicialización y *backend*.
- **/app/models/**: modelos de aprendizade automático que usa la aplicación.
- **/app/static/**: ficheros de contenido estático de la web, entre los que se incluye el contenido descargable, las imagenes y las hojas de estilo.
- **/app/templates/**: plantillas para generar el código html final.

- **/app/translations/**: archivos de localización para diferentes idiomas.
- **/app/uploads/**: directorio para almacenar los archivos cargados por los usuarios de la web.
- **/documentacion/**: documentación del proyecto.
- **/documentacion/tex/**: directorio con los archivos LaTex de la documentación.
- **/documentacion/img/**: imagenes usadas en la documentación.
- **/src/**: código fuente de los experimentos.
- **/src/datos/**: archivos con los datos de entrenamiento y test para los modelos.
- **/src/prototipos/**: código de los modelos y funciones necesarias para el procesado de los datos y el entrenamiento.
- **/src/prototipos/img/**: imagenes del desempeño de los modelos generadas durante el entrenamiento.
- **/src/prototipos/saved_models/**: modelos generados.

D.3. Compilación, instalación y ejecución del proyecto

Esta sección explica como replicar el entorno de trabajo para poder ejecutar cualquiera de los experimentos llevados a cabo así como la forma de ejecutar la aplicación web, ya sea de forma directa mediante el código fuente o traves de un contenedor Docker.

Git

Para obtener el código fuente del repositorio es necesario contar un cliente Git. Basta con instalar la versión adecuada al sistema operativo de la computadora desde la pagina web [3].

Para clonar el repositorio habría que navegar hasta el directorio elegido y ejecutar el siguiente comando en la consola:

```
git clone
https://gitlab.com/HP-SCDS/Observatorio/2019-2020/ubu-exoplanetia.git
```

Anaconda

La forma más rápida y sencilla de replicar el entorno para poder trabajar es utilizar Anaconda. La versión individual es gratuita y puede ser descargada desde su página web [1]. Para este proyecto se ha usado Python 3.7, por lo que es necesario instalar la versión de Anaconda correspondiente. No es necesario instalar Python de forma separada, ya que el instalador de Anaconda lo incluye.

Tras la instalación, el siguiente paso sería crear el entorno con las librerías necesarias y sus dependencias. Esta información se encuentra recogida en el archivo *conda_environment.yml* que se puede localizar en el directorio raiz del proyecto. Para la creación del entorno basta ejecutar el comando:

```
conda env create -f conda_environment.yml
```

Por defecto, el entorno será creado con el nombre **ExoplanetIA**. Si se desea usar un nombre distinto, este puede ser cambiando editando manualmente el archivo.

Finalmente, debemos activar el entorno creado:

```
conda activate ExoplanetIA
```

En este punto ya podemos Jupyter para consultar o ejecutar los notebooks creados o crear uno nuevo.

Web app

En el código del proyecto se incluye una aplicación web que permite ejecutar los modelos. La web puede ser desplegada en un servidor online (actualmente se encuentra disponible en Heroku) o ejecutarse en modo local. En este caso, es necesario navegar hasta el directorio *app* y ejecutar el siguiente comando:

```
python runsite.py
```

Esto iniciará el servidor y la web estará disponible para aceptar peticiones en la dirección *127.0.0.1:5000*

Docker

Es también posible descargar la aplicación web en formato de contenedor Docker, lo que nos permite ejecutarla en cualquier computador sin tener que instalar todas las librerías necesarias para el proyecto o incluso sin tener que descargar el código fuente.

El primer paso es decargar el contenedor al equipo. Como se comentó anteriormente, hay contenedores disponibles para cada release (a partir de la versión 1.2) y una con los últimos cambios del repositorio, denominada *latest*. Para descargarla, es necesario usar el comando *docker pull*. En caso de querer descargar, por ejemplo, la versión 1.2, usariamos el siguiente comando:

```
docker pull
registry.gitlab.com/hp-scds/observatorio/2019-2020/ubu-exoplanetia:1.2
```

Una vez descargado el contenedor, debemos arrancarlo, cosa que haremos con el comando:

```
docker run -dp 5000:5000
registry.gitlab.com/hp-scds/observatorio/2019-2020/ubu-exoplanetia:1.2
```

Una vez ejecutado, la aplicación web estará nuevamente lista en la url *127.0.0.1:5000*.

D.4. Manual del programador

Describimos a continuación como otro programador puede usar el proyecto para realizar nuevos experimentos sobre el codigo ya existente. Para ello, es requisito tener disponible el código fuente y el entorno replicado, procesos descritos en la sección anterior. En cualquier caso, este anexo no pretende ser un manual de uso de Python, Jupyter o cualquier otra de las librerías usadas en este proyecto y se da por sentado que el programador tiene conocimientos de programación y experiencia con dichas herramientas.

El proceso general consta de cuatro pasos:

- Cargar los datos
- Procesar los datos
- Definir el modelo
- Entrenar el modelo

Como se ha comentado, el conjunto de datos originales del proyecto se encuentran en la ruta `/src/datos/` y, por organización del código, es recomendable que cualquier otro conjunto de datos que se aporte se almacene en dicho directorio. A la hora de cargar los datos, este proyecto no presenta nuevas utilidades; la librería Pandas permite la lectura y carga en memoria

de archivos csv invocando solamente una función, por lo que es el metodo recomendado. Es importante notar que el conjunto de datos de origen, los valores de la columna *LABEL*, que determinan si hay o no exoplanetas en dicha estrella, estan codificados con los valores 2 y 1 para los casos de que exista o no exoplaneta. Aunque es posible trabajar con estos valores, es mucho más práctico cambiarlos y usar en su lugar 1 y 0. En el siguiente fragmento de código se puede observar como, tras declarar algunas constantes relativas a la ruta y los nombres de los archivos de datos, se leen y se modifican los valores tal y como se ha descrito.

```
PATH = "../datos/"
TRAIN_FILE = "exoTrain.csv"
TEST_FILE = "exoTest.csv"
```

```
#Leemos los datos de ambos datasets
df_train_raw = pd.read_csv(f'{PATH}{TRAIN_FILE}', low_memory=False)
df_train_raw['LABEL'] = df_train_raw['LABEL'].add(-1) #pasamos de 1-2 a 0-1
```

Figura D.1: Lectura del conjunto de datos

Aunque los datos pueden ser usados directamente para entrenar un modelo, como hemos visto, esto no da buenos resultados, siendo necesarios procesarlos. El programador tiene libertad aquí para aplicar los métodos que considere más eficaces o que mejor sirvan a sus objetivos, aunque algunos, como la normalización, son practicamente obligatorios para conseguir resultados aceptables.

En nuestro modulo de utilidades hemos implementado varias de ellas que el programador puede utilizar a su conveniencia. Estas son: normalización mínimo-máximo (`min_max_scaling`), normalización de puntuación z (`z_score_normalizing`), filtro gaussiano, tanto en su forma de suavizado como de reductor de la señal original (`gaussian_filter`), reducción de picos de intensidad (`reduce_upper_outliers`) y transformación de Fourier (`fourier_transformation`). A continuación podemos ver un ejemplo del procesado de datos en este caso aplicamos en primer lugar una reducción de los picos de intensidad, posteriormente un filtro gaussiano y terminamos normalizando.

```
df_train.iloc[:,1:] = reduce_upper_outliers(df_train.iloc[:,1:])
df_train.iloc[:,1:] = df_train.iloc[:,1:].apply(gaussian_filter, axis = 1)
df_train.iloc[:,1:] = df_train.iloc[:,1:].apply(z_score_normalizing, axis = 1)
```

Figura D.2: Procesado de los datos

A la hora de definir el modelo el programador puede usar alguno de los dos proporcionados en este proyecto: el perceptrón multicapa o la red LSTM. Ambos estan definidos en sendas clases (`modelo_perceptron` y `modelo_lstm`), por lo que solo deben ser instanciadas según necesidad. Ambas permiten cierta configuración; en el caso del perceptrón, el número de neuronas de las capas y, en el caso de la red LSTM, el número de neuronas, el número de capas y el porcentaje de neuronas que se anularan en cada ejecución (*dropout*).

El programador también debe seleccionar un algoritmo optimizador y una función de perdida de entre las soportadas por Pytorch. Hay que tener en cuenta que el uso de alguna de estas funciones condiciona el número de neuronas de nuestro modelo, especialmente los de la capa de salida. Por ejemplo, mientras que la función *CrossEntropyLoss* admite que la salida de nuestra red sean dos o mas neuronas, la función *BCEWithLogitsLoss* nos obliga a usar solamente una neurona como salida.

En la siguiente imagen podemos observa la instanciación de un modelo de perceptrón en el que se utiliza un array de enteros para especificar el número de neuronas de sus capas. Asimismo, se observa que nuestras clases soportan el uso de la función de Pytorch (*to*), que permite que un modelo sea cargado en la CPU o en la GPU, en caso de disponer de ella, permitiendo hacer uso de sus características para acelerar el entrenamiento. Por último, vemos que en este ejemplo se esta usando la función *CrossEntropyLoss* y el algoritmo *SGD*.

```
sizes = [1598, 1000, 250, 20]
modelo = Perceptron(sizes).to(device)
criterion = torch.nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(modelo.parameters(), lr = learning_rate)
```

Figura D.3: Definición de un modelo de perceptrón fijando el número de neuronas de sus capas

El último paso para finalizar un experimento consiste en entrenar el modelo, para lo que debemos definir ciertos parámetros. El número de epochs,

la tasa de aprendizaje y el nombre del modelo son obligatorios, mientras que alpha, beta y device son opcionales. Device nos permite especificar si nuestro modelo se cargará y entrenará en la CPU (el valor por defecto) o en la GPU, mientras que alpha y beta indican los coeficientes para ajustar nuestra función de valoración del modelo. Una mayor alpha dara más peso a la sensibilidad mientras que incrementar beta dará más importancia a la especificidad. Por defecto, ambas toman el valor de 0.50.

```
epochs = 100
learning_rate = 0.001
alpha = 0.5
beta = 0.5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_name = os.getcwd() + "\\saved_models\\perceptron_sgd_cross_fourier"
```

```
resultado = train_cross(modelo, model_name, criterion, optimizer, epochs, alpha, beta,
                        df_train_fourier, df_validation_fourier, device)
```

Figura D.4: Parametrización y entrenamiento del modelo

En nuestro codigo se incluyen tres diferentes funciones para entrenar modelos, `train_cross`, `train_bce` y `train_lstm`. Las dos primeras son usadas a la hora de entrenar modelos basados en el percetrón, una cuando se utilizan funciones de coste como *CrossEntropyLoss*, que admiten redes con varias neuronas de salida y la otra para redes que solo presentan una neurona de salida y usan funciones como *BCEWithLogitsLoss*. Finalmente, la tercera se utiliza para entrenar modelos basados en LSTM. El código de estas funciones es muy similar y será refactorizado en futuras versiones, dejando solamente una función general para entrenar cualquier modelo.

Las tres funciones de entrenamiento devuelven un objeto de la clase `resultados`. Este objeto almacena las puntaciones obtenidas por el modelo durante el entrenamiento, permitiendo mostrar las gráficas correspondientes y guardar en disco las imagenes generadas.

```
: resultado.plot_graphics()
```

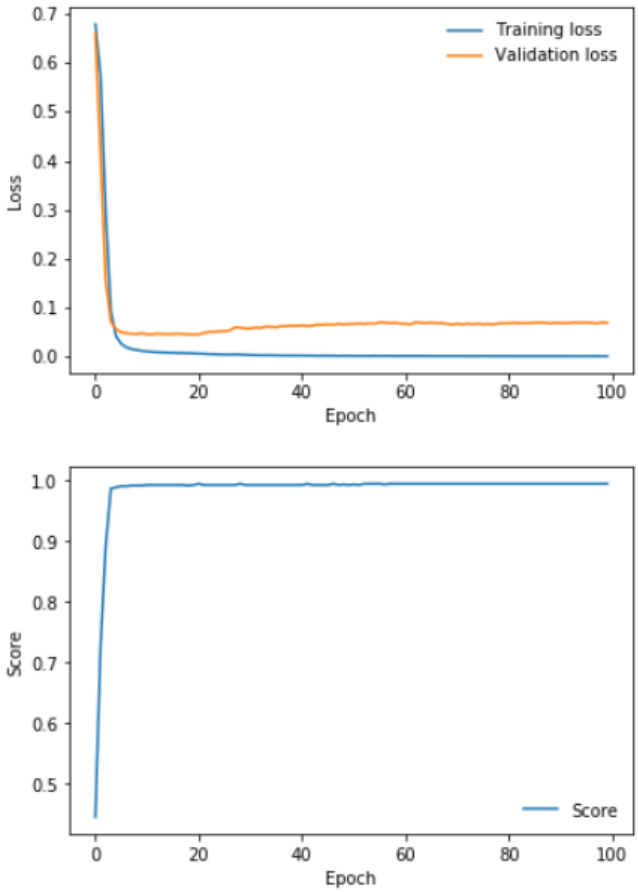

Figura D.5: Gráficas generadas tras el entrenamiento del modelo

Apéndice E

Documentación de usuario

E.1. Introducción

Aunque este proyecto siempre ha estado dirigido hacia la investigación y el desarrollo de modelos de aprendizaje automático y no hacía el desarrollo de una aplicación para usuarios finales, si se ha implementado una página web que permite ejecutar modelos y mostrar los resultados. En este apartado explicaremos las formas de instalarla y usarla.

E.2. Requisitos de usuarios

Actualmente es posible utilizar la aplicación web de formas distintas, instalandola directamente los archivos en un servidor web o utilizando Docker para ejecutar el contenedor.

A la hora de utilizar directamente la versión web es necesario:

- Sistema operativo linux o similar, de 32 o 64 bits.
- 2 GB RAM.
- Python 3.
- 500 MB libres en el disco duro.

En lo que se refiere a la instalación via Docker, necesitamos tener instalada dicha aplicación en el sistema. En su página web no hay especificados requisitos mínimos para su instalación en sistemas unix, pero si para sistemas

Windows, donde es necesario la aplicación Docker Desktop. En este caso, es necesario [4]:

- Windows 10 64 bit, versión Pro, Enterprise o Education.
- 4 GB RAM.
- Tener habilitadas las características de Hyper-V y Containers Windows.

Los pasos necesarios para la instalación pueden encontrarse igualmente en la web de Docker [4].

Además de los requisitos de Docker, es necesario contar con 500 MB extra libres en el disco duro para almacenar el contenedor.

E.3. Instalación

Los procesos de instalación son, en ambos casos, sencillos. En el caso de querer instalar la aplicación, los pasos a seguir son:

- Copiar el contendido de la carpeta */app/* al directorio seleccionado del servidor.
- Copiar los archivos *Procfile* y *requirements.txt* al mismo directorio.
- Abrir una terminal y navegar a dicho directorio donde debemos ejecutar el siguiente comando para instalar los paquetes necesarios:
  `pip install -r requirements.txt`
- Arrancar la aplicación ejecutando el comando:
  `python runsite.py`

En el caso de realizar via Docker, como se comentó en el apartado previo, debemos primero descargar el contenedor al equipo:

```
docker pull
registry.gitlab.com/hp-scds/observatorio/2019-2020/ubu-exoplanetia:1.2
```

Y, una vez descargado el contenedor, arrancarlo:

```
docker run -dp 5000:5000
registry.gitlab.com/hp-scds/observatorio/2019-2020/ubu-exoplanetia:1.2
```

En ambos casos, la aplicación web estará ejecutandose en la url interna *127.0.0.1:5000*.

E.4. Manual del usuario

El contenido del sitio es mayormente estático, paginas webs que solo presentan texto y con las que no es posible interactuar. El acceso a estas páginas se realiza a traves de los enlaces provistos en la cabecera, donde también se encuentra un botón para cambiar el idioma del sitio. Actualmente, el sitio esta disponible en español y inglés.

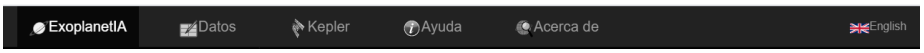

Figura E.1: Cabecera de la web con los diferentes enlaces

Actualmente el sitio cuenta con dos modelos distintos para probar, uno basado en redes LSTM y otro en el perceptrón multicapa. Es posible seleccionar el modelo a traves de un desplegable en la página inicial. A la hora de proporcionar los datos, es posible descargar desde el propio sitio un pequeño archivo con ejemplos. Este archivo se encuentra localizado en la ruta */static/downloads/minitest.csv*, aunque se proporcionan accesos a su descarga desde la página inicial y la página de datos. Además, es posible también usar los dataset originales disponibles en la web del proyecto[2] en Kaggle.

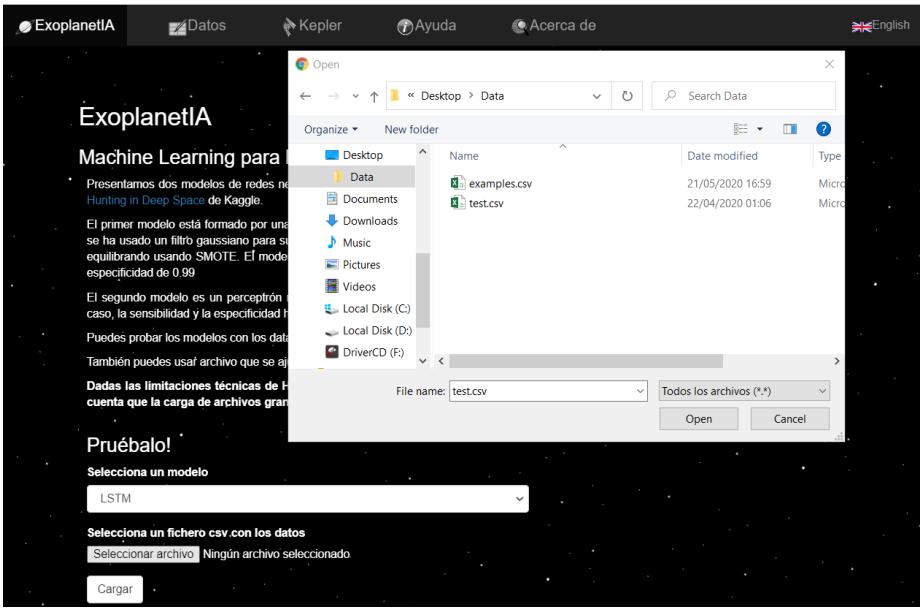

Figura E.2: Seleccionando un archivo para cargar

Una vez seleccionado el archivo y tras pulsar el boton *Cargar*, el fichero es subido al servidor y procesado. El usuario será redirigido a la página de resultados. Aquí es posible ver la cantidad de estrellas en las que se han encontrado exoplanetas así como los índices de dichas estrellas en el archivo de datos. Además, se muestra una gráfica con la cantidad de estrellas con exoplanetas encontradas respecto al total. Y, finalmente, para cada estrella con exoplaneta encontrada, se muestran dos gráficas, una mostrando la intensidad del flujo de luz y la otra la intensidad.

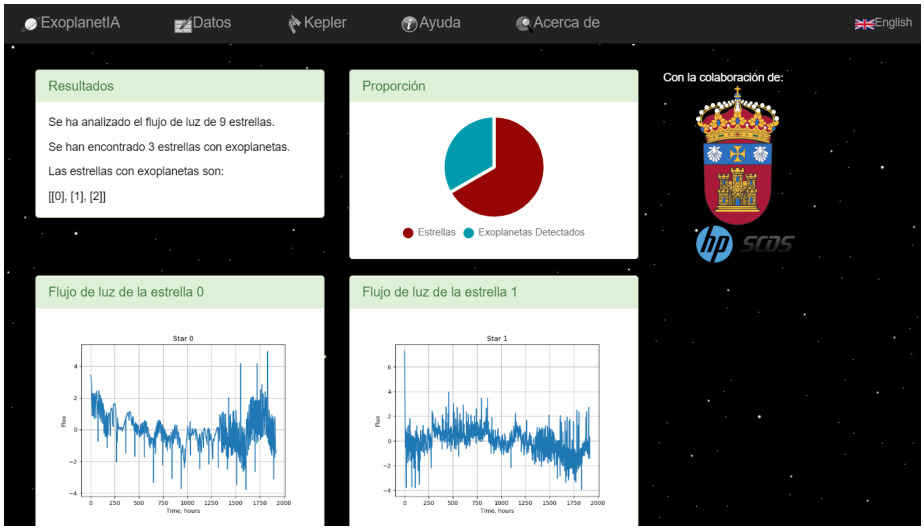

Figura E.3: Resultado de analizar el fichero

Bibliografía

[1] Anaconda. https://www.anaconda.com/products/individual. Accessed: 21/06/2020.

[2] Exoplanet hunting in deep space. https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data. Accessed: 14/05/2020.

[3] Git. https://git-scm.com/downloads. Accessed: 21/06/2020.

[4] Install docker desktop on windows. https://docs.docker.com/docker-for-windows/install/. Accessed: 15/06/2020.

[5] K2 mission. https://archive.stsci.edu/k2/. Accessed: 21/05/2020.