UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

ExoplanetIA
Machine Learning para la detección de exoplanetas

Presentado por Jesús María Herruzo Luque
en Universidad de Burgos — 31 de mayo de 2020
Tutor: Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Jesús María Herruzo Luque, con DNI 44372813V, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 31 de mayo de 2020

Vº. Bº. del Tutor:

D. Carlos López Nozal

Vº. Bº. del co-tutor:

D. Manuel Hermán Capitán

Vº. Bº. del co-tutor:

D. Alejandro Viloria Lanero

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

Introducción

Desde hace miles de años, el ser humano ha contemplado las estrellas en el cielo nocturno, sintiéndose fascinado por esa multitud de puntos luminosos. Primero construyó leyendas a su alrededor, mitos con los que intentaba comprender su realidad y dar significado a su mundo, pero pronto observó que algunas de esas estrellas no estaban quietas en la bóveda celeste, sino que se movían, realizando complejos círculos que trataron de entender, estudiar y predecir. Sin saber aún lo que eran, estaban observando otros planetas.

La observación del cielo y de los cuerpos que lo habitan ha sido una actividad continuada durante toda la historia del ser humano. Con el paso de los siglos y el desarrollo de nuevas ideas y tecnologías, nuestro conocimiento del cosmos no ha parado de crecer, primero en nuestro entorno *cercano*, para ir, en los últimas décadas, adentrándose en regiones cada vez más alejadas de nuestro sistema solar. Estrellas y planetas han perdido su carácter místico, dando lugar a teorías sobre su formación, su ciclo vital y su muerte.

Aún así, la mayor parte del universo permanece desconocido. Sólo en nuestra galaxia, la Vía Láctea, se estima que existen entre 100.000 y 400.000 millones de estrellas y, según los últimos datos, el número de galaxias estimado en el universo observable es de unos dos billones. Algunas estimaciones sobre el número de planetas, según consideremos su número medio por estrella, lo sitúan en torno a 10^{25}. Y, sin embargo, no fue hasta 1995, con el descubrimiento de Dimidium, que tuvimos constancia del primer exoplaneta.

Aunque el número de exoplanetas descubiertos ha ido creciendo con los años, la pregunta es obvia, ¿cómo es, entonces, que habiendo tantos , conocemos tan pocos? Y, más importante aún, ¿podemos hacer algo para detectar más exoplanetas y hacerlo de forma más rápida? En este trabajo estudiaremos técnicas que intentarán responder dicha pregunta.

1.1. Kepler y la detección de exoplanetas

Kepler es un telescopio espacial lanzado por la NASA el 7 de marzo de 2009. Nombrado así en honor al astrónomo alemán Johannes Kepler y colocado en órbita heliocéntrica, el objetivo de la misión era buscar planetas extra solares, especialmente aquellos de un tamaño similar a la Tierra, situados en la zona de habitabilidad de su estrella.

La detección de exoplanetas de forma directa, esto es, observándolos directamente mediante un telescopio, es una tarea muy complicada, siendo muy pocos los descubiertos de esta forma. Ello se debe principalmente a que los planetas no emiten luz propia, sino que simplemente reflejan parte de la luz de sus estrellas. Siendo, además, muy pequeños en comparación con su estrella, es fácil que su brillo quede eclipsado por el de su estrella madre. Así pues, los planetas detectados de esta forma suelen tener dos características comunes: son gigantes gaseosos muy alejados de su estrella.

Sin embargo, es posible detectar exoplanetas de formas indirectas. Una de ellas, la usada por el telescopio Kepler, es el conocido método del tránsito. En términos astronómicos, un tránsito ocurre cada vez que un objeto pasa por delante de otro mayor, bloqueando su visión. El ejemplo más directo es un eclipse solar, durante el cual la Luna se coloca entre la Tierra y el Sol, bloqueando de forma total o parcial la visión de este. De la misma forma, si estuviésemos observando cualquier estrella y un planeta pasase por delante de ella, notaríamos una disminución en la intensidad de su luz.

Dotado de un sensible fotómetro y con un campo de visión fijo, Kepler fue apuntando hacia las constelaciones del Cisne, Lira y Dragón para captar de forma simultánea la luz emitida por unas 150.000 estrellas. La duración inicial de la misión estaba prevista en tres años y medio, pero el ruido generado en los datos estaba haciendo mayor de lo esperado, lo que hacía necesario un mayor tiempo para completar sus objetivos. El plazo de la misión fue extendido hasta 2016, pero el infortunio hizo que dos de los giroscopios se estropearan, uno a mediados del 2012 y otro en mayo del 2013. Con solo otros dos giroscopios operativos, la misión original tuvo que ser abandona. En su lugar, tras escuchar diferentes alternativas por parte de la comunidad científica, la NASA aprobó la nueva misión de Kepler, denominada K2, Segunda Luz.

El 30 de octubre del 2018, Kepler agotó totalmente su combustible y fue apagado por la NASA. Durante sus nueve años de servicio, Kepler observó 530.506 estrellas y descubrió 2.662 exoplanetas, aproximadamente un 70 % de todos los que conocemos. Sus datos nos permitieron estimar que en sólo

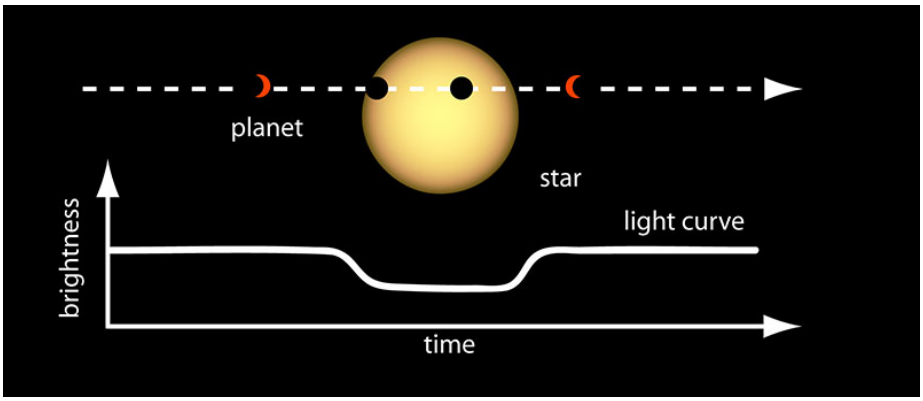

Figura 1.1: Disminución del flujo de luz durante el transito planetario [6]

en la Vía Láctea existen por lo menos otros 17.000 millones de exoplanetas de tamaño similar al nuestro. Muchos de estos datos aún siguen estudiándose; enterrados en esos datos hay otros 2900 planetas aún sin confirmar. Pero la búsqueda no termina aquí: el 18 de abril del mismo año, la NASA lanzaba el telescopio TESS para continuar la detección de nuevos mundos usando, igualmente, el método del tránsito. Y en 2021 está previsto el lanzamiento del telescopio James Webb con la tarea de examinar los hallazgos más prometedores de Kepler y TESS. ¿Cuantos nuevos planetas nos ayudaran a descubrir?

1.2. Machine learning y la busqueda de exoplanetas

Dada la evidente cantidad de información obtenida por el telescopio Kepler así como su complejidad, es obvia la necesidad de automatizar el proceso de la búsqueda de exoplanetas. Para ello se han utilizado diferentes algoritmos como VARTOOLS [11] o Transit Least Squares (TLS) [12], ambos basados en estudiar los periódicos picos de caída de la intensidad de la luz [15].

Dado que la heterogeneidad de los datos hace bastante complejo su estudio con un enfoque algorítmico tradicional, ha surgido otro enfoque distinto, basado en técnicas de aprendizaje automático. Se han propuesto diversos modelos para encontrar soluciones adecuadas, caracterizadas principalmente por el tipo de arquitectura de red utilizada. Así, podemos encontrar soluciones basadas en arboles de decisión, perceptrones multicapa,

redes recurrentes, convolucionales o varias de ellas mezcladas. Pero también se han propuesto alternativas que buscan estudiar no las variaciones del flujo de luz, sino su frecuencia. En cualquier caso, esto no quita que estas aproximaciones no adolezcan, también, de otras tantas dificultades. En entre ellas, resaltan dos: la gran cantidad de ruido en los datos y la escasez de ejemplos de exoplanetas confirmados.

En este trabajo intentaremos encontrar soluciones mediante el aprendizaje automático. Para ello construiremos modelos con diferentes arquitecturas y realizaremos análisis tanto del flujo de luz como de su frecuencia. Trabajaremos, además, con técnicas para reducir el nivel de ruido de los datos y estudiaremos si es posible mejorar nuestros modelos mediante la generación automática de ejemplos positivos adicionales.

Objetivos del proyecto

- Investigar las técnicas y soluciones existentes de aprendizaje automático para la detección de exoplanetas mediante el análisis del flujo de luz.
- Comparar diversas librerías de aprendizaje automático para, en base a sus características, elegír la más adecuada para implementar modelos con diferentes arquitecturas e hiperparámetros.
- 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.
- Comparar nuestro modelo con otros presentados en la plataforma de Kaggle.
- Desarrollar una pequeña aplicación web que permita ejecutar el modelo seleccionado. La aplicación debe permitir cargar un fichero con datos de flujos de luz, analizarlos y presentar los resultados.
- Utilizar un sistema de control de versiones para gestionar los cambios en el código.
- Utilizar una metodología ágil para el desarrollo y la planificación del proyecto.

Conceptos teóricos

3.1. Minería de datos

La minería de datos es el proceso de extraer información de grandes conjuntos de datos usando para ello técnicas de diversos campos, como la estadística, el aprendizaje automático o los sistemas de bases de datos. Aunque no es nueva, su uso durante los últimos años ha crecido de forma drástica. Varios factores explican este incremento. Por un lado, tenemos factores que han permitido generar enorme cantidad de datos, como la fuerte penetración de Internet en casi todo el mundo o la reducción del precio de los chips, permitiendo su incorporación a practicamente cualquier objeto y convirtiendolos, de facto, en ubícuos. Por el otro lado, tenemos la reducción del precio de almacenamiento. En 1956, el precio de un GB de disco duro tradicional costaba más de nueve millones de dólares. A día de hoy, ese precio se ha reducidos a menos dos centavos [3].

Ahora bien, convertir toda esa cantidad de datos en información de la que se pueda extraer conocimiento no es tarea fácil, existiendo distintos métodos y metodologías para obtener resultados. Uno de estos métodos es CRISP-DM [2], *Cross Industry Standard for Data Mining*, codificado en 1996 por un grupo de interés especial constituido tanto por empresas proveedoras de herramientas (SPSS y Teradata) cómo por empresas usuarias (Daimler, NCR y OHRA).

Este método se divide en seis pasos:

- Entender el negocio, esto es, entender los objetivos y los requisitos del proyecto.

- Entender los datos, que incluye la recogida de los datos, así como su exploración inicial para detectar problemas de calidad o subconjuntos para formular hipótesis.
- Preparar los datos para solventar cualquier problema detectado en la fase previa y facilitar su procesado por los modelos.
- Modelar, donde se definen las técnicas y arquitecturas que se van a usar y se construyen los modelos. Desde este punto es posible volver a la fase previa de preparación de los datos, ya que diferentes modelos pueden necesitar diferentes técnicas de preprocesado de los datos.
- Evaluar los modelos generados y seleccionar finalmente uno o varios de ellos, en función de las necesidades.
- Desplegar el modelo seleccionado de forma que pueda ser usado para realizar predicciones sobre nuevos datos.

Veamos a continuación como se aplican las diferentes etapas del proceso CRISP-DM en nuestro proyecto.

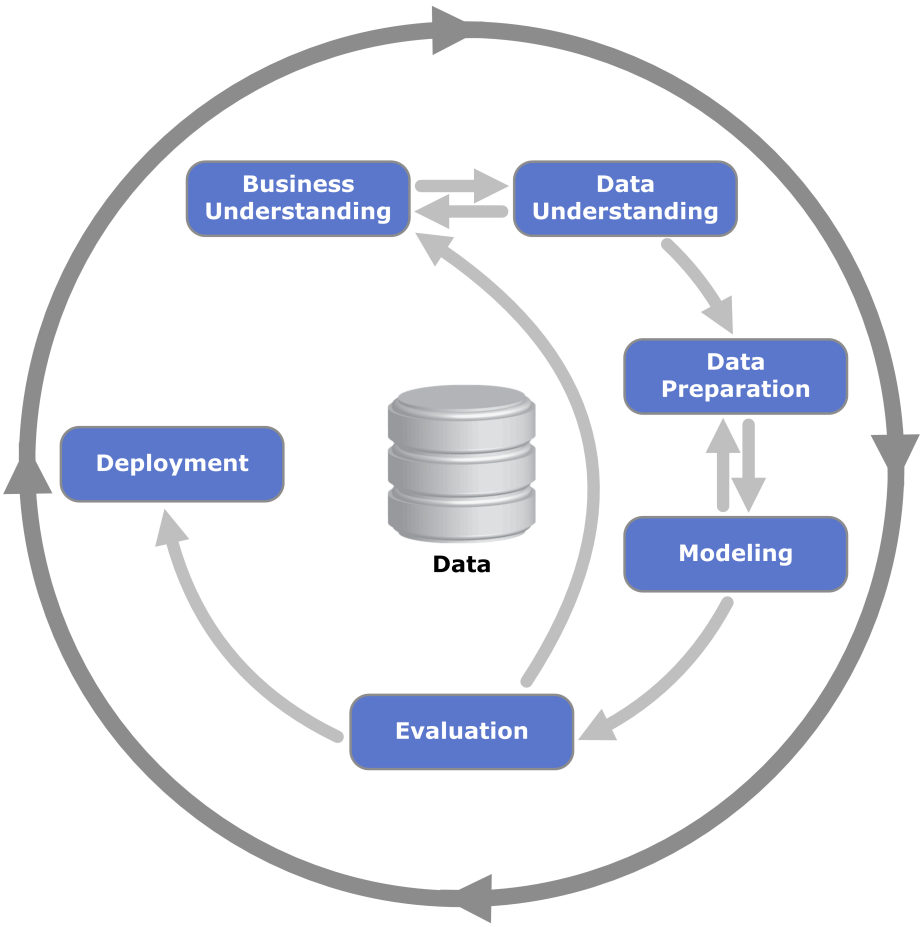

Figura 3.2: Fases del método CRISP-DM [2]

Entendiendo el negocio: objetivos y requisitos

Ya hemos comentado en el capítulo previo los Objetivos del proyecto, por lo que no los repitiremos aquí.

Entendiendo los datos

Los datos para este proyecto han sido descargados de Kaggle, donde podemos encontrar su descripcion: "los datos describen el cambio en el flujo (intensidad de luz) de varios miles de estrellas. Cada estrella tiene una etiqueta binaria de 2 o 1. Un 2 indica que se confirmó que la estrella tiene al menos un exoplaneta en órbita; algunas observaciones son, de hecho, sistemas de varios planetas."[4]

La fuente original de los datos pertenecen a la misión Kepler de la NASA, concretamente a la campaña 3 de la fase K2. Se pueden encontrar en la web del Mikulski Archive Mikulski Archive [5]. Los datos originales han sido transformados por Dan Lessmann; el proceso se encuentra detallado en su repositorio en GitHub. Finalmente, los datos han sido incorporados a Kaggle, desde donde han sido descargados.

Nuestro dataset se compone de dos archivos, uno con los datos de entrenamiento y otro con los datos para testear. Su composición es la siguiente:

- Fichero de entrenamiento:

  - 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

- Fichero de test:

  - 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

Lo primero que se puede observar es que un dataset enormemente desbalanceado. Las estrellas con exoplanetas confirmados solo representan el 0.73 % en el dataset de entrenamiento y el 0.88 % en el de test, lo que representa un grave handicap para el entrenamiento exitoso de los modelos. Veremos posteriormente técnicas para lidiar con este problema.

Vamos a proceder a mostrar gráficamente algunos ejemplos de estrellas con y sin exoplanetas para intentar captar características comunes o posibles anomalías.

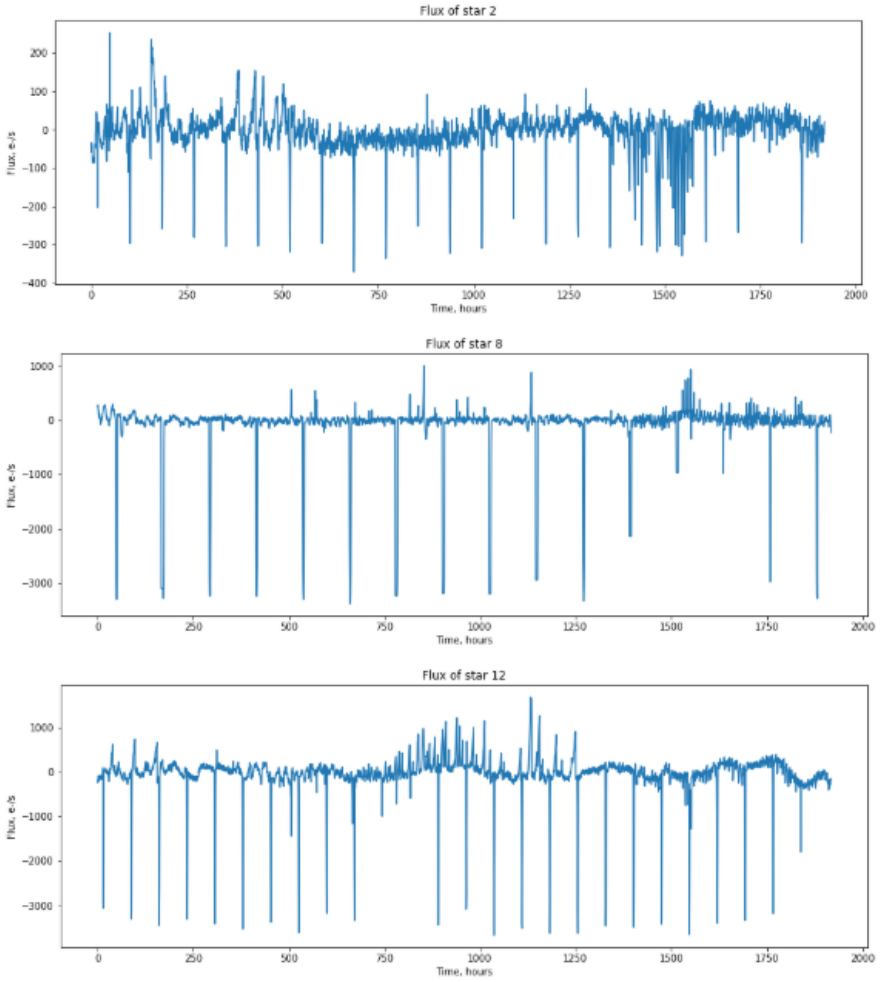

Figura 3.3: Estrellas con exoplanetas confirmados

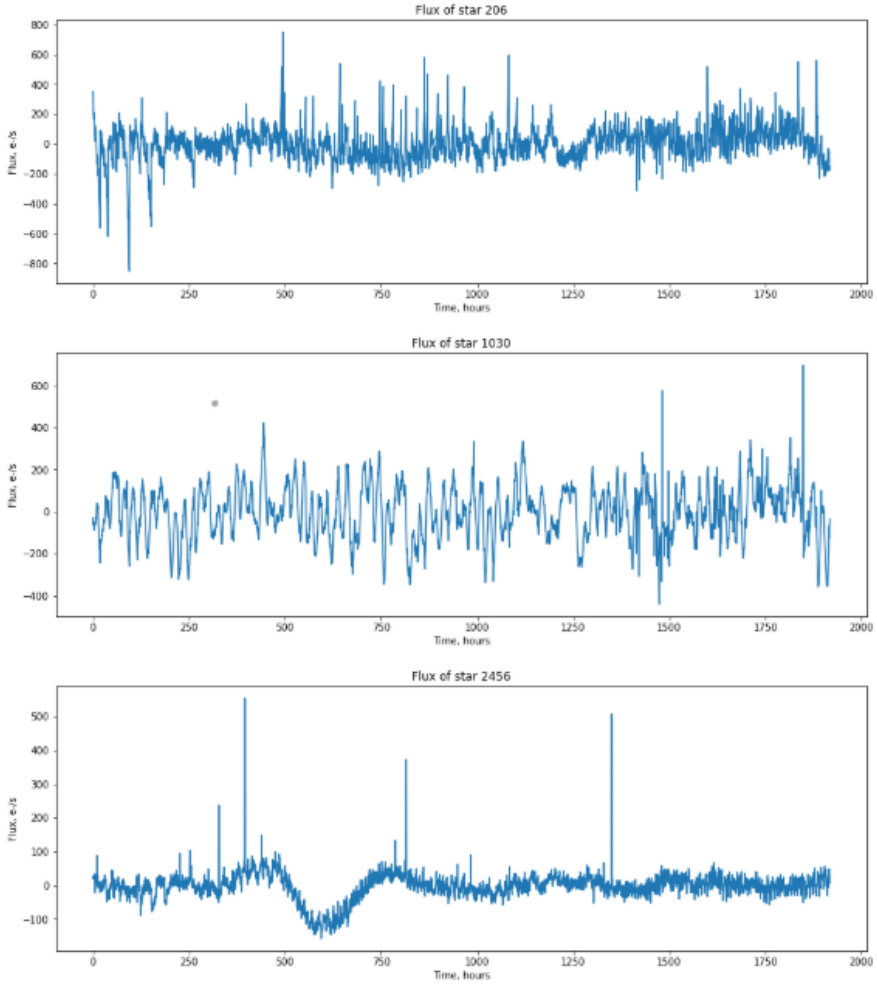

Figura 3.4: Estrellas sin exoplanetas confirmados

Observando las imágenes destacan dos cosas. Por un lado, la amplitud en el rango de la intensidad de luz y, por otro, la presencia puntual de picos de luz. Esta es una circunstancia extraña, ya que el brillo de una estrella suele permanecer relativamente estable. Y, desde luego, no es debido al transito de una planeta, ya que esto reduciria el valor, como se aprecia en los picos descendentes de las primeras graficas. Asi pues, lo mas probable es que estos valores sean errores y lo mejor será descartarlos durante el procesado de los datos.

Preparación de los datos

Rara vez los datos se recogen de forma que puedan ser evaluados de forma directa por un modelo, por lo que necesitan de un tratamiento previo que haga más factible extraer información de ellos. Por supuesto, no todos los datos van a ser tratados de idéntica forma y, dependiendo del estado y del objetivo del proyecto, habrá que utilizar unas técnicas u otras, aunque hay algunas cuyo uso está tan extendido que su uso se ha convertido en un paso obligatorio de casí cualquier proyecto.

Una de estás técnicas es la normalización, cuyo objetivo es cambiar los valores de las columnas numéricas de forma que usen una escala común. Esto nos permitirá abordar uno de los problemas detectados anteriormente, la amplitud en el rango de valores de nuestros datos.

Además de la normalización, veremos más adelante otra serie de técnicas que nos ayudaran a entrenar el modelo, tales como *undersampling* y *oversampling*, uso de filtros para suavizar la señal o el estudio de la frecuencia.

Modelado

En esta fase del proceso es cuando se definen los modelos y arquitecturas que se usaran. Nosotros nos centraremos especialmente en el uso del Perceptrón Multicapa y, en menor medida, las redes LSTM, un tipo de redes neuronales recurrente.

Evaluación

Una vez tenemos los modelos entrenados, es el momento de evaluarlos y seleccionar el mejor. La diferencia de objetivos y de características de los datos, puede llevar a emplear métricas muy diversas para evaluarlos. En el caso que nos atañe, donde pretendemos construir un clasificador binario, nos centraremos especialmente en evaluar la sensibilidad y especificidad de

nuestro modelo. Entraremos en más detalle en Medidas de desempeño del modelo.

Despliegue

El paso final del proceso incluy la puesta en producción del modelo elegido de forma que pueda usarse para obtener predicciones reales sobre nuevos datos. Dependiendo del proyecto, algunas opciones habituales incluyen despliegues en aplicaciones webs, aplicaciones móviles o su contenerización.

3.2. Normalización

Como hemos comentado anteriormente, normalizar consiste en transformar los valores numéricos de una o más columnas para que sus valores se ajusten a una escala común y es más importante cuanto más amplio sea el rango de valores que adopta el dataset.

Normalizar no es un proceso único, sino que existen diversas técnicas para normalizar, cada una con ventajas y desventajas asociadas por lo que, en función de los datos, puede ser conveniente usar una u otra. Las dos más usadas son la normalización mínimo-máximo y la puntuación z. La primera de ellas tiene la ventaja de transformar todos los valores al rango [0, 1] siendo sensible a valores atípicos extremos mientras que la puntuación z, menos sensibles a estos valores extremos, no garantiza un rango fijo.

$$minmax = \frac{X - min(x)}{max(x) - min(x)}$$

$$zscore = \frac{X - mean(x)}{stdev(x)}$$

3.3. Filtro gaussiano

Dada la elevada sensibilidad del satélite Kepler a la intensidad de luz y, como se observa en las imagenes previas, las señales presentan numerosos picos y valles que pueden dificultar el aprendizaje de los modelos. Para solucionar este problema, usaremos un filtro gaussiano, un efecto de suavizado que puede aplicarse a una señal o imagen para reducir el ruido.

A la hora de aplicarlo tenemos dos opciones distintas. La primera es aplicar el filtro tal cúal, provocando el suavizado la señal, quitandole ruido, pero también nitidez y detalles; detalles que pueden ser importantes. La otra alternativa consiste reducir la señal original con el resultado de aplicar

el filtro gaussiano, consiguiendo de esta forma reducir la importancia de la señal base y resaltando los detalles más especificos.

3.4. Undersampling y Oversampling

Cuando, como es nuestro caso, tenemos un dataset fuertemente desbalanceado, es necesario aplicar técnicas que permitan el correcto entrenamiento de la red. Dos de ellas son *undersampling* y *oversampling*.

La primera de ellas consiste en reducir el tamaño de nuestro dataset, normalmente hasta que tengamos el mismo número de ejemplos de cada clase, aunque pueden usarse otras proporciones. La técnica habitual consiste en seleccionar de forma aleatoria elementos de clase mayoritaria y eliminarlos. Una ventaja derivada de este método es que, al tener que procesar un menor número de elemtnos, el entrenamiento es más rapido. Entre las desventajas tenemos, en primer lugar, que al privar a la red de determinados ejemplos le estamos ocultando información que puede ser importante y provocar que la red no generalize bien. De forma similar, cuando el número de ejemplos de una clase es muy pequeño, como es nuestro caso, el dataset puede ser demasiado pequeño, causando que la red memorice los datos y sea incapaz de generalizar.

La otra técnica, *oversampling*, consiste en el proceso opuesto, esto es, generar nuevos elementos de clase minoritaria. Igualmente, se suelen generar ejemplos hasta conseguir la igualdad, aunque no siempre es el caso. A la hora de generar los nuevos ejemplos, el enfoque más sencillo y rápido consiste en realizar copias de los datos de la clase minoritaria. La desventaja evidente de este proceso es que la red no incorpora nueva información y, al procesar los mismos datos una y otra vez, es más probable que acabe memorizandolos y, por tanto, sin capacidad de generalizar ante datos nuevos.

Para tratar de solucionar este problema existen diversos métodos, siendo uno de los más utilizados SMOTE [9] (*synthetic minority oversampling technique*). Aunque existen multiples variantes, la forma básica para generar nuevos elementos consiste en, para cada elemento de la clase minoritaria, se seleccionan sus K (generalmente, 5) vecinos más próximos. De entre ellos, se selecciona uno al azar y se generan una más instancias (dependiendo del número total de instancias a generar) en la recta que une las instancias originales.

3.5. Transformada de Fourier

La transformada de Fourier es una transformación matemática que permite descomponer una función, generalmente expresada en función del tiempo, en sus frecuencias constitutivas. En nuestro caso, el estudio de la frecuencia puede ser interesante dado que la existencia de uno o más planetas orbitando la estrella resultaría en la existencia de más de una frecuencias.

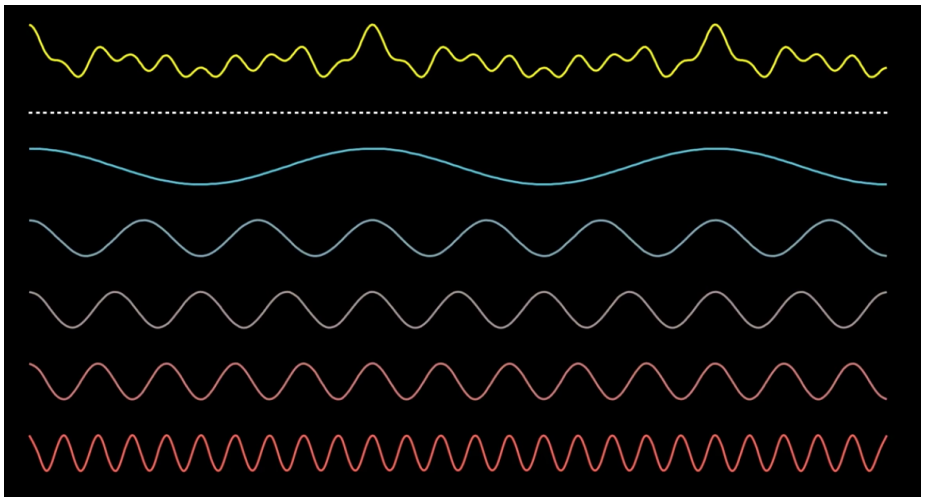

Figura 3.5: Ejemplo de una señal (arriba, en amarillo) y su descomposición en las frecuencias que la componen [1]

El nombre de transformada de Fourier, en honor al matemático francés Joseph Fourier, hace referencia tanto a la representación como a la función que la produce. Es, además, una operación reversible, permitiendo pasar de un dominio a otro.

El análisis matemático de la transformada de Fourier está fuera del alcance de este trabajo, pero es posible encontrarlo, junto con información más detallada en diferentes fuentes como, por ejemplo, la wikipedia [7].

3.6. Perceptrón Multicapa

El perceptrón multicapa, o red multicapa con propagación hacia delante, es el modelo de aprendizaje profundo por excelencia. Es una generalización del perceptrón simple que surgió debido a la incapacidad de estos para dar solución a problemas no lineales [15].

El objetivo de estas redes es aproximar alguna función f. Por ejemplo, para un clasificador como es nuestro caso, `y = f(x)` mapea un input, x a una categoría y. La red define un mapeado `y = f(x,` θ`)` y aprende los valores de los parametros θ que resultan en la mejor aproximación a la función [10].

Su arquitectura es simple, consistiendo en una capa de entrada, encargada de recibir las señales del exterior y propagarlas a las neuronas de la siguiente capa, una o más capas ocultas, que procesan la información, aplicando una función de activación a los datos recibidos de la capa previa, y una capa de salida, que comunica al exterior la respuesta de la red.

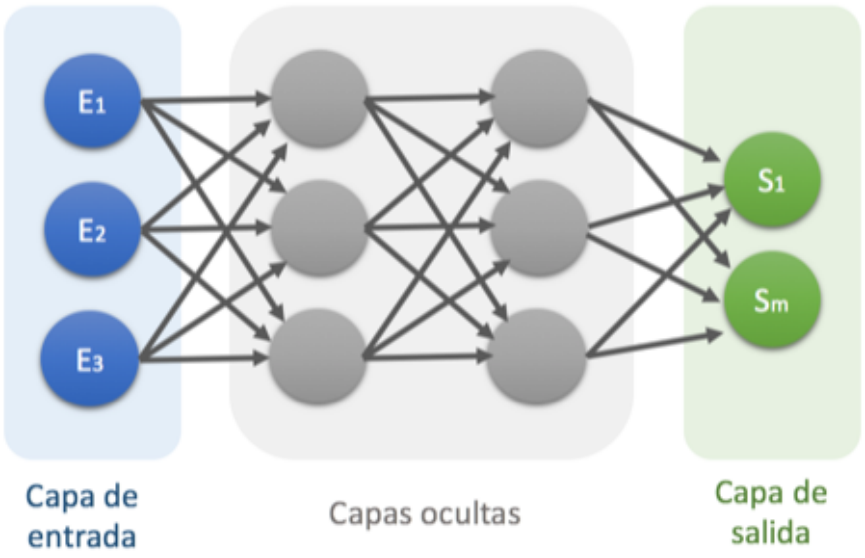

Figura 3.6: Perceptrón multicapa

Consideraciones de diseño

A la hora de diseñar la arquitectura de un perceptrón multicapa hay varios elementos que podemos alterar para tratar de lograr una mejor solución.

Número de neuronas

En algunos casos, especialmente en la capa de entrada y de salida, el número de neuronas viene definido por el problema a resolver. En las capas ocultas, este número puede variar ampliamente. Hay que considerar que

un número elevado de neuronas puede provocar que estas memorizen los datos de entrada, proceso conocido como *overfitting*. En este caso, nuestra red proporcionaría buenos resulados durante con los datos entrenamiento, pero sería incapaz de generalizar y los resultados serian pobres cuando se enfrentase a datos nuevos.

Por otro lado, un número escasos de neuronas puede provocar el efecto contrario, que nuestra red no disponga de la capacidad necesaria para generalizar correctamente. En este caso, conocido como *underfitting*, la red presenta pobres resultados, tanto en el entrenamiento como en los tests.

En nuestro caso, el número de neuronas en la capa de entrada viene determinado, a priori, por el número de características de nuestro dataset, esto es, 3197. Respecto a la capa de salida, dependerá de la función de activación que se vaya usar; en el problema que tratamos de resolver, clasificando los datos en dos categorías, hay o no hay exoplaneta, usaremos dos neuronas.

Número de capas y conexiones

De forma similar al número de neuronas, el número de capas ocultas puede variar ampliamente, contribuyendo especialmente al problema de *overfitting* comentado anteriormente. Además, de acuerdo al teorema de aproximación universal, cuando se usan funciones de activación no lineales, una sola capa oculta es suficiente para representar cualquier función continua en un rango dado, aunque esta capa puede ser demasiado grande y fallar en aprender y generalizar correctamente [10], por lo que es conveniente probar varias aproximaciones. Dado que nuestro problema es, además, no continuo, no debemos ceñirnos a usar una sola capa oculta.

Es también importante como se encuentran conectadas las capas. El modelo es más frecuente es el de capa totalmente conectada, donde cada neurona esta conectada a cada una de las neuronas de la siguiente capa. Hay, sin embargo, otras opciones donde, la más frecuente de ellas, consiste en que la salida de algunas o todas las neuronas de una capa se conectan con la entrada de neuronas de una capa no inmediatamente posterior, haciendo que su valor tenga más peso en el resultado final de la red.

Otro opción respecto a las conexiones entre las capas es usar la técnica conocida como *dropout*. Esta consiste en asignar, durante el proceso de entrenamiento, el peso de determinadas neuronas, seleccionadas de forma aleatoria, a cero, excluyendo de facto su aportación al resultado final de la red. El objetivo de la técnica es reducir el *overfitting*, ya que hace que la red sea menos dependiente del peso especifico de determinadas neuronas [16].

Funciones de activación

Vamos a examinar brevemente las funciones de activación más frecuentes que podríamos usar en nuestro modelo.

La función sigmoide fué de las primeras en usarse de forma masiva. La función está acotada entre [0, 1] y suele usarse en la última capa para representar probabilidades en clasificadores binarios. También es habitual usarla en las capas ocultas de los perceptrones multicapa. Sin embargo, adolece de algunos problemas, quizá el mayor de ellos sea que satura y mata el gradiente, provocando una lenta convergencia.

La tangente hiperbólica es muy similar a la sigmoide, estando igualmente acotada, aunque en un rango mayor, [-1, 1], lo que la hace adecuada para problemas en los que hay que decidir entre dos opciones. A diferencia de la sigmoide, esta centrada en el 0.

Es importante resaltar que la función sigmoide y la tangente hiperbólica se encuentran relacionadas, tal que $tanh(x) = 2 * sigmoid(2x) - 1$ por loq que existe poca diferencia a la hora de usar una u otra.

Tenemos también la función relu, función lineal rectificada por sus siglas en inglés. Esta función no está acotada y deja los valores positivos sin alterar pero transforma los negativos en cero. Tiene un buen desempeño en redes convolucionales a la hora de tratar con imagenes, pero también es la opción por defecto en los perceptrones multicapa. Esto se debe principalmente a dos factores: es poco probable que mate el gradiente y genera redes escasas, esto es, redes con neuronas muertas que no se activan, lo que hace la red más eficiente. Además, su facilidad de cálculo respecto a otras funciones, acorta el tiempo de entrenamiento de la re. Presenta, sin embargo, un importante problema, y es que puede matar a demasiadas neuronas. Para solucionarlo, existe una variante, denominada leaky relu, que, en lugar de anular los valores negativos, los multiplica por un coeficiente para devolver un valor negativo.

Finalmente, hablamos de la funcion softmax, que transforma un vector de entrada en un vector de probabilidades cuyo sumatorio es uno. Es usada frecuentemente en la capa de salida de la red cuando se trata de resolver un problema de clasificación.

De cara al diseño de nuestro modelo, la elección evidente para la capa de salida es usar una función softmax, aunque también se realizarán pruebas usando la función sigmoide para ver si presenta un mejor resultado.

Respecto a las capas ocultas, usaremos principalmente la función relu y, de forma similar a con la capa de salida, haremos pruebas con la sigmoide.

Algoritmo de optimización

El algoritmo de optimización es el encargado de actualizar los pesos de nuestra red para minimizar la perdida. Pytorch ya tiene implementados varios de estos algoritmos, por lo que solo queda decidir cual usar.

El más conocido y uno de los primeros en desarrollarse es el descenso del gradiente. Pytorch implementa el descenso del gradiente estocástico (SGD), que actualiza los pesos tras procesar cada ejemplo, en lugar de hacerlo tras procesar todo el dataset. Este algoritmo presenta algunas dificultades, como elegir la tasa de aprendizaje adecuada y que ese valor se aplique a todos los pesos por igual, así como oscilaciones que dificultan la convergencia en el punto mínimo. Para intentar corregir este último problema, el algoritmo puede configurarse para usar momento, que ayuda a suavizar las oscilaciones añadiendo una fracción de los pasos previos al paso actual.

Usaremos este optimizador como linea base de trabajo con diferentes valores para la tasa de aprendizaje tanto con como sin momento.

Otro de los algoritmos más usados es Adam [13], acrónimo en inglés de estimación adaptativa del momento, que será el otro algoritmo que usaremos.

A diferencia de SGD, Adam calcula tasas de aprendizaje distintas para los parámetros e incorpora momento. Adam es computacionalmente eficiente, tiene pocos requisitos de memoria y facilita la convergencia. Además, al actualizar los parámetros con diferentes tasas de aprendizaje, es menos sensible a una elección no óptima de la tasa de aprendizaje inicial.

3.7. LSTM

Un tipo particular de redes neuronales son las recurrentes, que permiten añadir la dimensión temporal al procesado de datos. Esto lo consiguen incluyendo bucles en su arquitectura, permitiendo de esta forma la persistencia de determinada información y dotandolas de una gran facilidad para tratar información contenida en forma de secuencias o listas, donde un acontecimiento esta estrechamente vinculado a acontecimientos previos. Esto las ha convertido en el estandar para tratar diversos tipos de problemas como, por ejemplo, el reconocimiento de voz, el modelado del lenguaje o la traducción entre idiomas.

Para nuestro problema particular, este tipo de red puede resultar útil, ya que el transito planetario incluye una sencuencia concreta de pasos, donde el flujo de luz se muestra relativamente constante, posteriormente experimenta un decrecimiento hasta llegar un mínimo y finalmente vuelve a su nivel original. Además, si el periodo orbital del planeta es breve, podríamos encontrar repeticiones de esta secuencia. El procesado de este conjunto de datos temporales es la especialidad de las redes recurrentes.

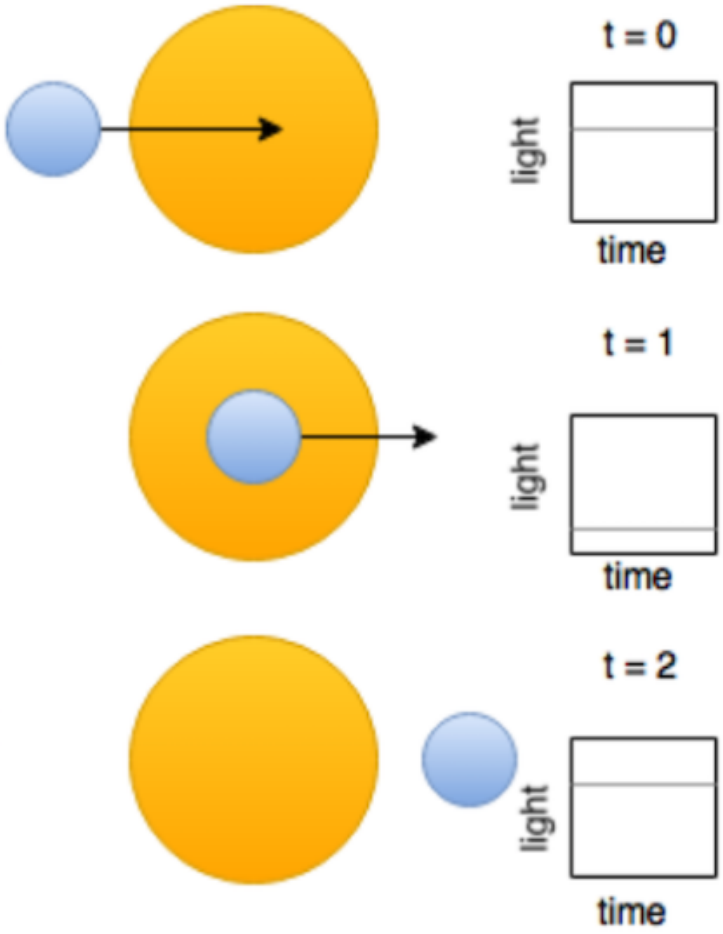

Figura 3.7: Variación de la intensidad de luz por tránsito planetario [4]

La forma más simple de conseguir esta recurrencia es conectando la salida de una neurona no solo hacia neuronas de las capas siguientes, como hemos visto en el caso del perceptrón multicapa, sino también hacia la entrada de la propia neurona o de neuronas de las capas previas. Esto es, la salida de

las neuronas de la capa h_t se encuentran conectadas a las neuronas de la capa h_{t-i}.

Esta configuración simple funciona muy bien para tratar secuencias donde la información se encuentra temporalmente próxima pero presenta problemas relacionando información distante en el tiempo o información que influye en muchos momentos pasados.

Pero existen muchos tipos de redes recursivas. En nuestro caso nos centraremos en las redes LSTM (*long short term memory*), uno de los estandares de la industria, y designadas con el objetivo de presentar memoria a largo plazo. La arquitectura básica de estas redes esta formada por capas LSTM, también llamadas células, cada una sirviendo de entrada a la célula siguiente. En su interior, cada una de estas células presenta los mismos elementos, siendo el principal C_t, el estado de la célula y la parte que proporciona la memoria a largo a plazo al sistema.

Otros elementos influyen sobre esta memoria. Una primera capa sigmoidal permite determinar que memoria a largo plazo mantener o desechar. Otras capas interiores, implementadas mediante funciones sigmoidales y tangenciales, permite determinar que información actual incorporar a la memoria a largo plazo. Finalmente, las capas finales se encargan de devolver la información procesada por la célula y de conectar su estado a la siguiente célula de la red.

Para una descripción más detallada del proceso, se puede consultar Understanding LSTMs.

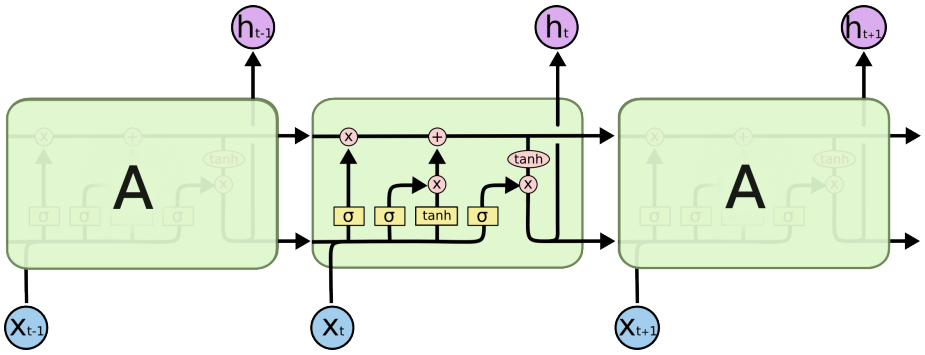

Figura 3.8: Detalle de una célula LSTM [8]

3.8. Medidas de desempeño del modelo

El objetivo de nuestro modelo es obtener un clasificador binario que nos indique la probabilidad de que una entrada de datos pertenezca a una de nuestras dos clases: exoplaneta y no exoplaneta.

Con este objetivo en mente, la literatura existente nos indica que la solución óptima suele ser aplicar una función de activación softmax para la capa de salida de la red. Esta función, también llamada función exponencial normalizada, es una forma de regresión logística que normaliza un valor de entrada en un vector de salida que sigue una distribución de probabilidad cuya suma total es 1. Así pues, el valor de salida de la neurona k-ésima vendrá dado por la función:

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

En cualquier caso, estudiaremos otras opciones, como puede ser el caso de la función sigmoide, que nos devuelve un valor en el rango [0, 1], el cual puede ser interpretrado como probabilidad, en nuestro caso, de que sea una estrella con exoplaneta.

Desbalanceo del dataset

Analizando nuestro conjunto de datos, observamos que este se encuentra muy desbalanceado: los casos negativos (no es un exoplaneta) superan ampliamente en número a los casos positivos (si es un exoplaneta).

Esto supone un problema para el aprendizaje de la red ya que, ante cualquier entrada, esta puede *"aprender"* a responder siempre que no es un exoplaneta, acertando en la amplia mayoría de los casos.

Para solventar este problema, siguiendo a Viloria [17], vamos a definir nuestra función de evaluación, con la que juzgaremos el aprendizaje real de nuestra red y su capacidad de predecir el resultado correcto frente a nuevas entradas de datos.

$$f = Acierto * (\alpha * Sen + \beta * Esp)$$

donde `Acierto` representa el ratio de respuestas correctas, `Sen` es la sensibilidad (casos catalagocados como positivos que son realmente positivos), `Esp` es la especificidad (casos negativos correctamente calificados como no exoplanetas) y α y β son dos pesos usados para alterar la importancia de la sensibilidad y la especificidad. Comenzaremos con un valor neutro de 0.5 para cada uno, pero estudiaremos si su ajuste permite obtener un mejor modelo.

Definimos a continuación los ratios de `Acierto`, `Sen` y `Esp`, donde `VP` es el número de verdaderos positivos, `VN` los verdaderos negativos, `FP` los falsos positivos y `FN` los falsos negativos.

$Acierto = \frac{VP+VN}{VP+VN+FP+FN}$

$Sen = \frac{VP}{VP+FN}$

$Esp = \frac{VN}{VN+FP}$

3.9. Bibliotecas de machine learning

De cara a implementar nuestro modelo de red neuronal debemos decidir que lenguajes y bibliotecas vamos a usar. A día de hoy, la mayor parte de los frameworks y bibliotecas que facilitan el desarrollo de redes neuronales funcionan en entornos Python, que puede ser considerado el lenguaje de facto de la industria, aunque existen otras alternativas en lenguajes como R, Mathlab, y en frameworks como Neuroph para Java o Mathematica.

En nuestro caso, vamos a considerar una comparativa de tres bibliotecas de Python:

- Tensorflow es una biblioteca de código abierta desarrollada por Google para uso interno, tanto en investigación como en producción, que posteriormente fue lanzada al público.

- Keras es una API de alto nivel capaz de ejecutarse sobre otros lenguajes o bibliotecas, como Tensorflow, R o Theano, diseñada con el foco en la facilidad de uso.

- PyTorch es una biblioteca de código abierta desarrollada principalmente por Facebook que también presenta una interfaz para C++.

Vamos a examinar diferentes parámetros para ver que nos aporta cada una de ellas.

Velocidad

Los estudios muestran que no hay una diferencia significativa de velocidad entre Tensorflow y PyTorch. Este no es el caso con Keras, que presenta un rendimiento claramente inferior.

Nivel del API

Como se ha comentado, Keras es una API de alto nivel, capaz de correr sobre otras bibliotecas, como Tensorflow o Theano, proporcionando una interfaz común que facilita el desarrollo rápido de proyectos.

Tensorflow proporciona APIs tanto de alto como de bajo nivel, lo que le dota una gran flexibilidad.

Finalmente, PyTorch proporciona sólamente una API de nivel, enfocada en el trabajo directo con matrices.

Arquitectura

Keras presenta una arquitectura simple y fácil de comprender, mientras que tanto Tensorflow como PyTorch presentan arquitecturas más complejas y un código con mayor verbosidad.

La API de PyTorch se encuentra mejor diseñada mientras que la de Tensorflow es un tanto confusa y ha recibido numerosos cambios importantes en cada versión, lo que dificulta mantener un código estable y estar actualizado.

Debuggin

Depurar código en Tensorflow es relativamente complejo y no muy intuitivo. En Keras no es un proceso habitual, dado el alto nivel de sus componentes, lo que tampoco facilita la depuración en caso de algún problema, ya que la mayor parte del código se encuentra en la biblioteca. Sin embargo, PyTorch si ofrece buenas opciones para la depuración, muy similares a las encontradas en IDEs para lenguajes conocidos, como Eclipse o Visual Studio.

Dataset

Los problemas de velocidad de Keras no lo hacen adecuado para trabajar con grandes datasets. No es el caso de Tensorflow o PyTorch, que no tienen este problema de rendimiento.

Documentación y comunidad

Tanto en PyTorch como en Tensorflow se nota el efecto de tener detras a dos grandes empresas tecnológicas. En ambos casos, existen númerosos recursos gratuitos con los que aprender así como una importante comunidad de usuarios que las respaldan y ofrecen su ayuda. Tensorflow tiene más

tiempo de desarrollo y su base de usuarios es mayor pero desde el 2018 la popularidad de PyTorch está en constante aumento, especialmente en el ambito académico.

Keras contrasta respecto a las otros dos con una más reducida comunidad y menor documentación.

Puesta en producción

A la hora de poner en producción un modelo previamente entrenado, Keras no dispone de ninguna utilidad en si misma, haciendo uso de las caracteristicas de Tensorflow. Este permite servir los modelos en un servidor web mediante una API REST o en dispositivos móviles.

PyTorch se apoya en otras bibliotecas para poder exponer sus modelos via web, permitiendo también otras opciones interesantes, como la interfaz con C++, lo que permite convertir los modelos en ejecutables fácilmente.

Resumen de la comparación

Tras analizar las características de las tres bibliotecas, vemos como se adaptan a nuestras necesidades.

Keras es una buena opción para probar y generar modelos de forma rápida, pero no nos permite profundizar en el aprendizaje y compresión de las redes neuronales, ya que la mayor parte del trabajo de nivel es gestionado de forma interna por la biblioteca. Unido a la dificultad de depurar el código y a su peor rendimiento, hace que optemos por no utilizarla.

La decisión entre Tensorflow y PyTorch es más dificil de realizar, ya que ambos aportan características similares: la posibilidad de trabajar con las redes a bajo nivel para poder estudiarlas en detalle, buen rendimiento y variados recursos para aprender, ya sea en forma de tutoriales o de comunidad de usuarios para resolver dudas. Sin embargo, hay dos detalles marcan la diferencia y nos hacen decantarnos por PyTorch: por un lado, la facilidad de depuración del código y, por otro, la posibilidad de generar ejecutables.

Así pues, la biblioteca que finalmente usaremos será **PyTorch**.

Técnicas y herramientas

Se exponen a continuación las herramientas usadas durante el desarrollo del proyecto.

4.1. Python

Python es un lenguaje de programación interpretado de alto nivel. Posee licencia de código abierto y, en los últimos años, se ha convertido en el estandar de facto para los proyectos de machine learning.

4.2. Jupyter Notebook

IDE interactivo de código abierto basado en web. Permite crear y compartir documentos, denominados notebooks, que contienen tanto código como texto markdown. Dichos notebooks serán nuestra herramienta principal de trabajo.

4.3. Bibliotecas de Pyhton

Torch

Biblioteca de código abierto para aprendizaje automático. Se puede encontrar más información sobre sus características y su comparación con otras bibliotecas similares en la Sección 3.9 Bibliotecas de machine learning.

NumPy

Biblioteca que agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

Pandas

Extensión de NumPy para manipulación y análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series de tiempo, permitiéndonos leer fácilmente los datos de los ficheros csv, así como manipularlos, aplicandoles diversas funciones o dividiendolos para formar los dataset de entrenamiento y validación.

SciPy

Basado en NumPy, expande esta biblioteca con herramientas y algoritmos para matemáticas, ciencias e ingeniería. Entre ellas se incluyen funciones para aplicar un filtro Gaussiano o realizar la transformada de Fourier. La primera de ellas la usaremos en preprocesado de datos para suavizar la señal mientras que la segunda será la base para entrenar los modelos en función de la frecuencia.

imbalanced-learn

Ofrece técnicas y algoritmos de *under-sampling* y *over-sampling* comúnmente utilizados en conjuntos de datos que muestran un fuerte desequilibrio entre clases. Nosotros solo usaremos una de sus funciones, SMOTE (Synthetic Minority Over-sampling Technique), para generar nuevos casos de estrellas con exoplanetas y así equilibrar el dataset.

Matplotlib

Matplotlib es una biblioteca para crear visualizaciones estáticas, animadas e interactivas en Python. Con ella mostraremos la evolución de nuestros modelos durante el entrenamiento mostrando como cambia la perdida, la puntuación o el área bajo la curva. Igualmente la usaremos para mostrar algunos ejemplos de nuestros dataset, ya sea en su forma original como flujo de luz o en su forma procesada, incluyendo su representación en forma de frecuencia mediante la transformada de Fourier.

Os

Proporciona una interfaz para utilizar los comandos del sistema operativo, como la navegación por directorios, que la usaremos para fijar la ruta donde guardar y, posteriormente, cargar, nuestros modelos.

Time

Aporta funcionalidades para trabajar con objetos de fechas y horas, permitiéndonos medir el tiempo que tarda el entrenamiento de los modelos.

HTML

Funcionalidades para trabajar con documentos HTML con el que podemos mostrar nuestra hoja de resultados, en HTML, dentro de un notebook de Jupyter.

Flask

Framework minimalista para crear aplicaciones web de forma rápida y sencilla.

Werkzeug

Biblioteca que proporciona diferentes utilidades relativas al desarrollo web. Nosotros vamos a usarla para validar el nombre del fichero que nuestra aplicación debe cargar y así evitar posibles ataques basados en el uso de carácteres especiales en el nombre de dicho fichero.

Wtforms

Añade representación y validación de formularios flexible para el desarrollo web con Python. Dado que nuestra aplicación web es muy simple, solo necesitamos un formulario para poder cargar un fichero conteniendo los datos a analizar.

Flask-wtf

Integra la biblioteca Wtforms con Flask.

Base64

Proporciona funcionalidades para codificar y decodificar datos binarios, permitiéndonos leer las gráficas generadas por Matplotlib y escribirlas en disco. De esta forma, después de que nuestra aplicación web analize el dataset proporcionado, puede generar las gráficas correspondientes y servirlas al usuario como parte de la página de resultados.

4.4. Git

Sistema de control de versiones distribuido de código abierto.

4.5. Gitlab

Servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Es una suite completa que permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas, ofreciendo una plataforma en la cual se puede realizar todas las etapas del ciclo de desarrollo del software.

4.6. LaTeX

Sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas. En nuestro caso, el sistema que usaremos para escribir la tesis.

MiKTeK

Distribución de LaTex para sistemas Windows.

TexMaker

Editor de código abierto de LaTex. Integra variedad de herramientas para desarrollar documentos.

Heroku

Heroku es una plataforma de computación en la nube, ofreciendo servicios de plataforma como servicio (PaaS). Desarrollada inicialmente para soportar solamente aplicaciones escritas en Ruby, hoy en día soporta muchos otros, como Scala, Node, Java o Python. Heroku ofrece un tier gratuito que usaremos para desplegar nuestra aplicación y así tener un ejemplo real en producción.

Aspectos relevantes del desarrollo del proyecto

5.1. Arranque del proyecto

Este proyecto comienza con la idea investigar y comparar los resultados de diversos modelos de aprendizaje para el problema de la detección de exoplanetas mediante la técnica del tránsito. Así mismo, se pretende estudiar y comparar los efectos que diversas técnicas de procesado de datos tienen en el resultado final.

Es de resañar que este problema, la detección de exoplanetas a traves del análisis del flujo de luz, es un problema ya resuelto, tanto con métodos algorítmicos tradicionales como con métodos de aprendizaje automático. Basta con un rápido vistazo a la página de Kaggle para encontrar numerosas y diversas soluciones, cuyo código puede ser copiado e implementado sin demasiados problemas. Por tanto, la creación final una aplicación para procesar nuevos datos no deja de ser un subproducto del objetivo principal de este proyecto, el estudio de diversos modelos y como sus características facilitan o entorpecen el análisis y la obtención de resultados.

Como hoja de ruta, estudiaremos un modelo simple de perceptrón multicapa y su rendimiento con los datos en bruto para, posteriormente, ir procesando los datos y observar los resultados. Posteriormente pasaremos a enfocar el problema desde otro punto de vista, analizando las frecuencias que componen la señal lumínica para, finalmente, trabajar con otra arquitectura de red neuronal, las LSTMs.

5.2. Perceptrón

Nuestra primera aproximación a la resolución del problema consiste en un perceptrón multicapa. El tamaño de nuestra capa de entrada viene determinado por la dimensionalidad de nuestro dataset, en nuestro caso, 3197 neuronas. Usaremos una capa oculta con 300 neuronas y finalmente una capa de salida con 2 neuronas que nos permitirá clasificar el resultado como estrella con o sin exoplanetas.

Como algoritmo de optimización vamos a usar SGD (*stochastic gradient descent*) y la función de perdida *CrossEntropyLoss*. En esta, intentaremos compensar el desbalanceo del dataset asignando valores distintos a los pesos de las clases, dando mayor importancia a la clase positiva, estrella con exoplanetas.

Esta primera aproximación, entrenada durante 50 epochs, no presenta buenos resultados. La función de perdida se estanca durante el entrenamiento, lo que significa que nuestra red ha dejado de aprender. Examinando los resultados, vemos se observa que devuelve, prácticamente con certeza total, que ninguna de las estrellas tiene exoplanetas. Esto se ajusta al resultado que esperabamos obtener dado el enorme desbalanceo del dataset: como apenas hay ejemplos de estrellas con exoplanetas, nuestra red ha aprendido que una forma de minimizar la función de perdida es dar siempre una respuesta negativa.

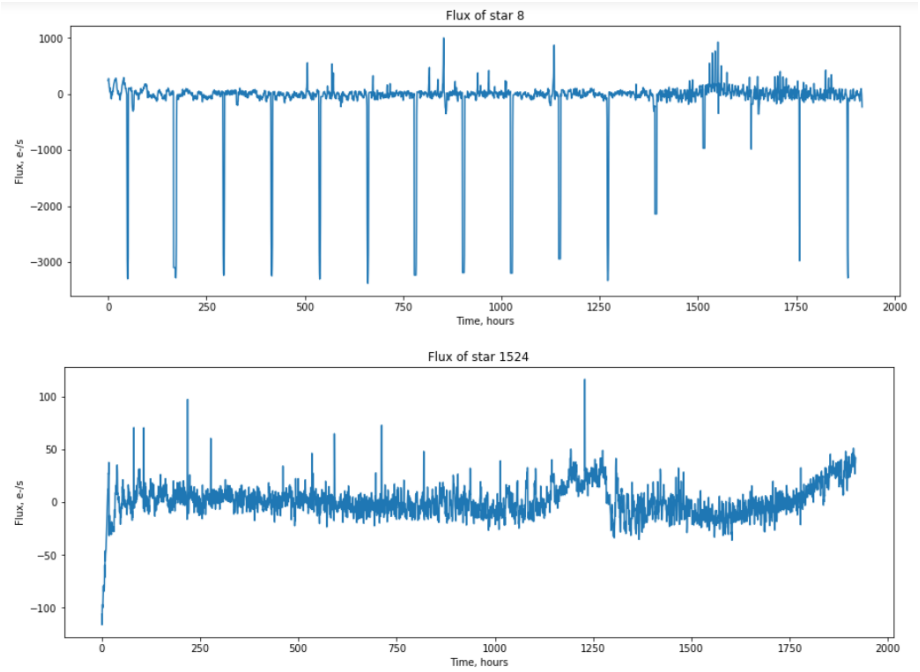

Figura 5.9: Flujo de una estrella con exoplanetas (8) y otra sin exoplanetas (1524)

El código correspondiente a este y a los siguientes experimentos puede encontrarse en el archivo *Perceptron_base.ipynb*.

Normalización y reducción de picos

Comenzamos ahora con el tratamiento de los datos para ver si podemos mejorar el resultado de nuestro modelo. El primer paso será reducir los picos de intensidad lumínica. A diferencia de la normalización, esta es una técnica concreta que usaremos dadas las características propias de nuestros datos. El razonamiento que nos lleva a usarla se basa en que el tránsito de un planeta frente a su estrella debe reducir el flujo, no aumentarlo, dado que los planetas no emiten luz propia. Así pues, ya sea por algún fenómeno físico propio de la estrella o simples errores de medición del satélite, lo que está claro es que cualquier pico de intensidad de luz no está relacionado con la existencia de exoplanetas.

Para lograr este objetivo usaremos la función *reduce_upper_outliers* definida en el archivo *utils.py*. El código ordena el valor del flujo y selecciona, en base a un porcentaje sobre el total, algunos de ellos. Posteriormente

calcula la media de los puntos a su alrededor y reduce los puntos señalados a dicho valor.

La otra técnica con la que vamos a trabajar, la normalización, es un elemento básico y habitual en la mayoría de los problemas de aprendizaje automático. Usaremos ahora la denominada *zcore*; como hemos comentado, este tipo de normalización no restringe los valores al rango [0,1] pero nos evita problemas cuando hay elementos que se distáncian mucho de la media. Observando la figura 5.9 Flujo de una estrella con exoplanetas (8) y otra sin exoplanetas (1524) se puede apreciar, especialmente en el caso de la estrella 8, que este es el caso en el que nos encontramos.

Comparando la figura previa con la figura 5.10 Flujo las estrellas 8 y 1524 tras reducir los picos y normalizar, se observa que los picos superiores de ambas estrellas se han eliminado. Asimismo, el rango de valores se ha reducido, moviendonos de la escala 10^3 a 10^1.

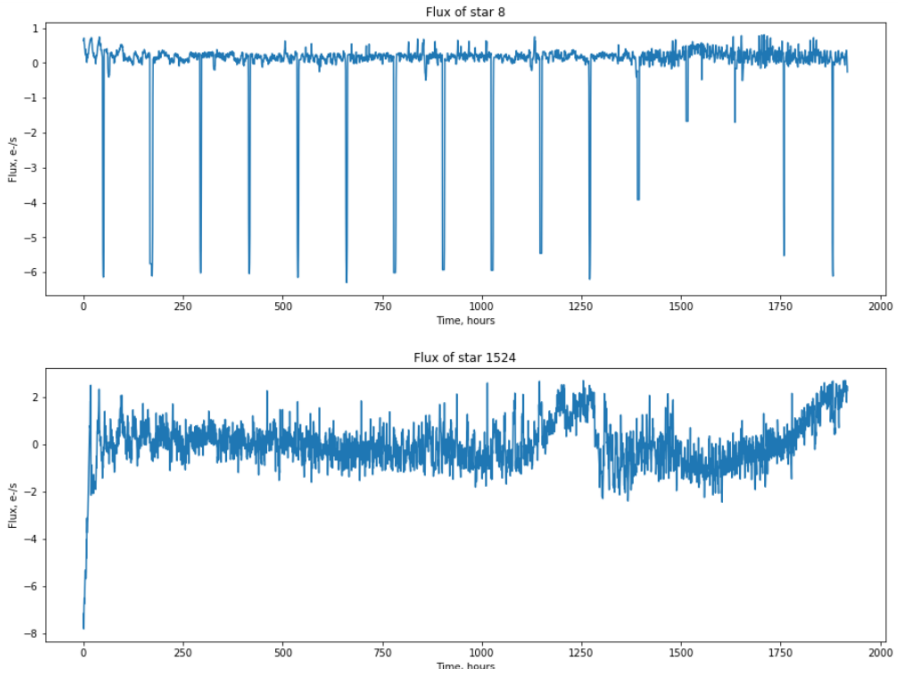

Figura 5.10: Flujo las estrellas 8 y 1524 tras reducir los picos y normalizar

En el entrenamiento de este modelo observamos que, igualmente, la red sigue sin aprender. Todas las estrellas son clasificadas como sin exoplanetas y la función de perdida decrece hasta estancarse hasta niveles muy bajos. Esto es, la red se ha estabilizado en certeza absoluta y va a devolver siempre que no hay exoplanetas.

Algoritmos de optimización y función de perdida

Procedemos a continuación a cambiar el algoritmo de optimización de nuestro modelo, pasando a utilizar Adam, tal y como comentamos en el apartado teórico Algoritmo de optimización. Los resultados siguen sin ser positivos. La perdida con el set de entrenamiento se estanca mientras que, con el set validación, aunque presenta picos de fluctuación, finalmente acaba también estancada.

Una de las diferencias más significas que encontramos en este punto son los distintos tiempos de entrenamiento. Mientras que el modelo usando SGD ha necesitado solamente 36 minutos para completar su entrenamiento, al usar Adam este tiempo se ha casí triplicado, pasando a ser de 95 minutos.

En este punto consideramos cambiar la función de perdida y probar con la función *BCEWithLogitsLoss (binary cross entropy with logits)* que, a diferencia de la primera, solo sirve como clasificador binario y solo necesita de la salida de una neurona. El resultado sigue siendo negativo, con una red que no aprende y, esta vez, con unas perdidas bastante mayores que en los casos anteriores.

Undersampling

Hemos comprado hasta ahora que las técnicas de normalizado de los datos y la eliminación de los picos de intensidad no suponen mejoría a la hora de entrenar nuestro modelo. Tanto con diferentes optimizadores como con diferentes función de perdida, el enorme desbalanceo de nuestros datos hace imposible obtener resultados positivos. Asi pues, vamos a intentar corregir ese problema y ver si podemos conseguir algún aprendizaje en la red.

Para ello, como discutimos en Undersampling y Oversampling vamos a proceder a reducir el tamaño de nuestro dataset. Siguiendo la técnica básica, creamos un nuevo dataset conteniendo todos los casos positivos (estrellas con exoplanetas confirmados) y seleccionando de forma aleatoria un número igual de casos negativos (estrellas sin exoplanetas). Esto nos deja con unos dataset muy reducidos, especialmente en el caso del set de validación (14 instancias solamente).

Aquí comenzamos ya a obtener algunos resultados positivos. La red comienza a discriminar y a clasificar realmente los datos en ambas categorías. En su mejor momento, tanto la sensibilidad como la especificidad alcanzan 0.857, una puntuación bastante buena.

Claro que no todo es positivo. El mejor resultado es obtenido en el tercer epoch del entrenamiento, tras lo cual la red diverge y la perdida en el set de validación se dispara. Además, hay una gran cantidad de datos negativos que la red no ha visto y que, en el futuro, podría facilmente catalogar como positivos. Es por ello que posiblmente este modelo, aunque presente buenos resultados en el entrenamiento, no sea capaz de generalizar correctamente en el futuro.

Filtro gaussiano

Otra técnica a nuestra disposición consiste en el Filtro gaussiano. Como hemos visto en las gráficas de intensidad de luz sin modificar, la señal presenta numerosos pequeños picos de subida y bajada debidos, principalmente, a la alta sensibilidad del fotómetro instalado en el satélite Kepler. Todas estas subidas y bajadas puede confundir a nuestra red, haciendo que considere que esta es la información importante que denota la existencia o no de exoplanetas.

Primero probamos a entrenar la red sustrayendo el resultado del filtro gaussiano a la señal original. Podemos ver la forma de la nueva señal en la figura 5.11 Resultado de sustraer el filtro gaussiano a la señal original. Para la estrella 8, con exoplaneta, vemos como se marcan más pronunciadamente las caidas de luz y como alrededor de ellas se levantan un par de picos intensidad. Esto puede marcar un buen camino para la red. Por el otro lado, en la estrella 2003, el filtro parece que solo ha aumentado el ruido, ampliando mas el rango de picos y valles, e incrementando la diferencia entre ambas estrellas.

Repecto al entrenamiento, observamos las perdidas, tanto en el set de entrenamiento como en el de validación, siguen un descenso estable hasta el epoch 80 aproximadamente, donde comienzan a diverger ligeramente. Es también sobre este punto cuando el modelo consigue sus mejores resultados, concretamente en el epoch 85, con una sensiblidad de 0.71, una especificidad de 0.99 y siendo el area bajo la curva de 0.86.

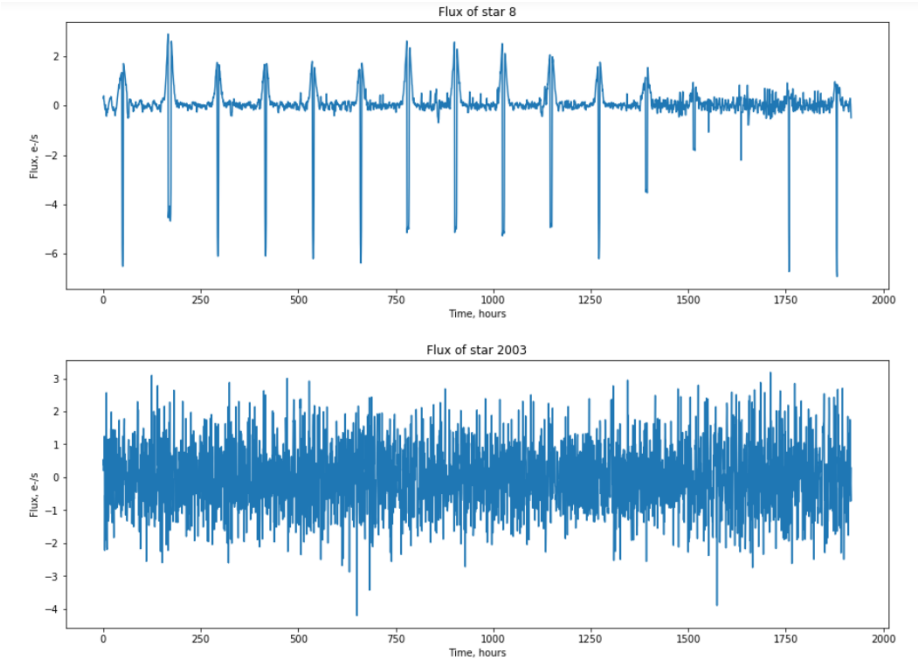

Figura 5.11: Resultado de sustraer el filtro gaussiano a la señal original

De forma similar, probamos a entrenar el mismo modelo, salvo que ahora usaremos el resultado del filtro gaussiano de forma directa, como nuestro dataset, en lugar de restarlo a la señal original como en el caso previo. El modelo alcanza una puntuación perfecta de forma rápida, en el epoch 31. Dado que no hemos marcado esto como condición de parada, el modelo sigue entrenando tratando de conseguir un ajuste más fino, pero las perdidas en el set de validación comienzan a oscilar, así como la puntuación del modelo. En cualquier caso, los resultados de este modelo parecen prometedores.

Repetimos nuestro análisis con ambas opciones para el filtro gaussiano, pero esta vez usamos Adam como algoritmo de optimización. En el primer caso, sustrayendo el filtro de la señal, obtenemos unos resultados interesantes. El área bajo la curva es de 0.93, con una sensibilidad de 0.86 y una especificidad de 0.99. Sin embargo, tanto estos valores como la perdida de validación fluctuan ampliamente. Usando solamente el filtro, el modelo consigue una puntuación perfecta relativamente pronto, en el epoch 22, manteniendo la perdida estable y baja. Hacia el final del entrenamiento vemos como la perdida de validación comienza a aumentar mientras que la perdida de entrenamiento permanece estable. En este punto nuestro modelo está comenzando a memorizar los datos del dataset de entrenamiento.

Por último, nos queda comprobar que resultados obtenemos al cambiar la función de perdida. Volvemos a entrenar nuevamente las dos variantes del modelo, usando en este caso la funcion *BCEWithLogitsLoss*. En este caso los resultados no son positivos para ninguno de los modelos. Todas las estrellas son clasificadas de forma negativa, nuestra red no esta consiguiendo aprender en estas condiciones.

SMOTE

Procedemos a continuación a probar una nueva técnica, el *oversampling*. Como discutimos en la sección de Undersampling y Oversampling vamos a trabajar con SMOTE. Aunque no siempre tiene que ser el caso, vamos a generar suficientes nuevas instancias como para igualar ambas clases, estrellas con y sin exoplanetas.

Existen numerosas variantes de SMOTE que pueden dar mejor resultado en función de los datos y del modelo de red. Sin embargo, como demuestra Kovács [14], los beneficios de usar alguna de estas variantes son menos significativas que las mejoras logradas por usar SMOTE frente a no usarlo. En base a ello, y dado que el uso de esta técnica increma de forma sustancial el tiemmpo de entrenamiento, vamos a proceder a usar solamente la forma original.

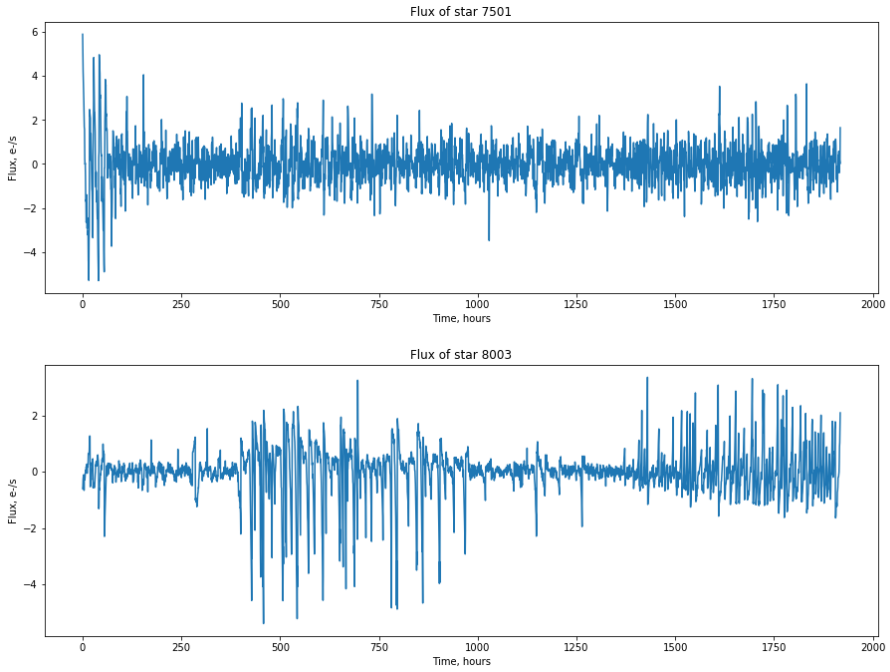

Figura 5.12: Nuevas instancias generadas mediante SMOTE

En el primer intento, usando *CrossEntropyLoss*, SGD y sustrayendo el filtro gaussiano de la señal obtenemos el mejor modelo en el epoch 19 con una sensibilidad de 1 y una especificidad de 0.99. Es, también, a partir de este punto que la perdida en validación comienza a aumentar mientras que en entrenamiento sigue descendiendo poco a poco. Es la situación habitual en la que el modelo se encuentra memorizando los datos y no aprendiendo. También vemos, como era de esperar, dado el mayor número de instancias, un incremento del tiempo de entrenamiento, pasando de los 45 minutos del modelo sin SMOTE a los 77 actuales. En el caso de usar solo el filtro gaussiano como dataset, la convergencia es aún más rapida, lograndose está en el epoch 7.

Cuando cambiamos el algoritmo de optimización y pasamos a usar Adam encontramos circunstancias similares. En ambos casos se produce una convergencia temprana y un, esperado, incremento en el tiempo de entrenamiento.

El código correspondiente a los experimentos con SMOTE puede encontrarse en el archivo *perceptron_smote.ipynb*

5.3. Análisis de frecuencia

Cambiamos el foco ahora para adentrarnos en el análisis de las frecuencias mediante la transformada de Fourier. El algoritmo ya se encuentra implementado en la libreria *SciPy* y será el que usaremos. Podemos observar en la figura Frecuencias de dos estrellas con exoplanetas confirmados el efecto de aplicar la transformada en los datos de un par de estrellas.

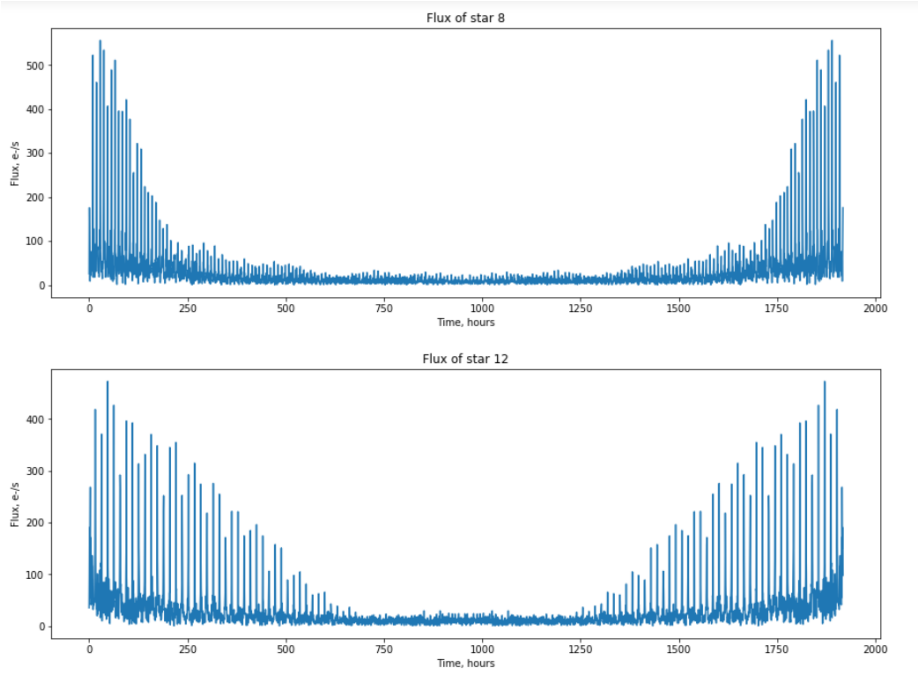

Figura 5.13: Frecuencias de dos estrellas con exoplanetas confirmados.

Es importante notar la simetría de la onda. Gracias a ello vamos a poder descartar la mitad de los datos, quedándonos solo con la primera parte. Esto podemos englobarlo dentro de las técnicas de reducción de la dimensionalidad, orientadas a reducir el número de características del conjunto de datos, eliminando las que se consideren superfluas, de forma que la red pueda centrarse en aquellas que mejor definen o catalogan los datos. Como beneficio secundario, al reducir el número de datos de entrada y reducir el tamaño de nuestra red para que sea concordante, es de esperar un tiempo menor de entrenamiento.

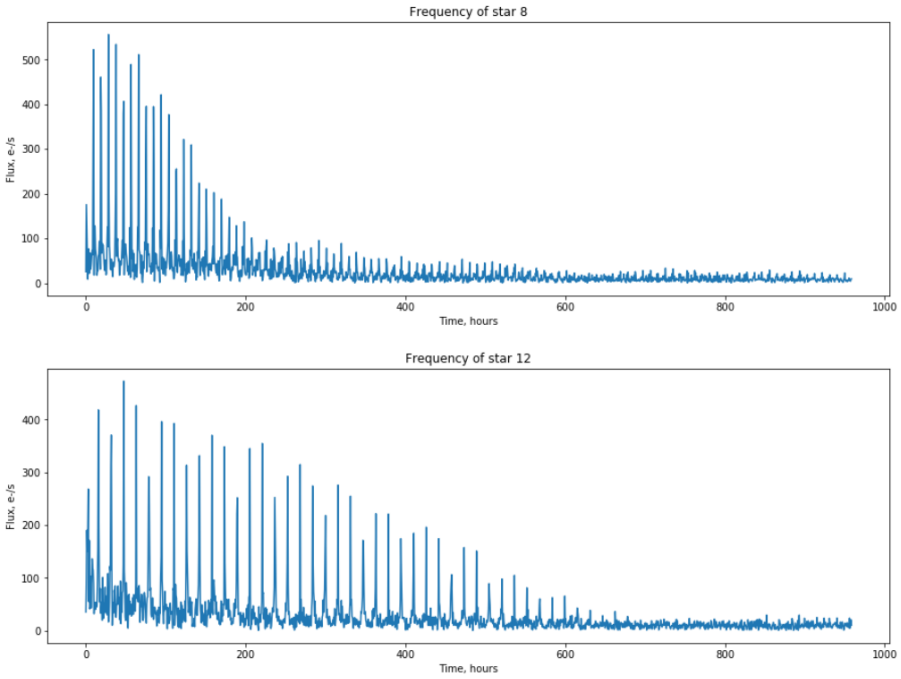

Figura 5.14: Señal reducida que usaremos en el entrenamiento.

Vamos a proceder a entrenar dos modelos usando el análisis de frecuencias. Para el primero usaremos el algoritmo SGD y la función *CrossEntropyLoss* junto con la reducción de picos y la normalización y, en el siguiente, incluiremos el uso de SMOTE para generar nuevas instancias.

El primero de los modelos obtiene unos resultados decentes, con una sensibilidad de 0.57 y una especificidad de 1 en el epoch 74. Aún así, vemos que la puntuación del modelo presenta grandes oscilaciones. La perdida en validación presena igualmente oscilaciones, con una tendencia claramente ascendente, aunque la perdida en el entrenamiento desciende. Por el contrario, al añadir SMOTE en el segundo caso, observamos una rápida convergencia ya en el epoch 4. La puntuación del modelo apenas oscila y las perdidas se mantienen bajas, por lo que parece un modelo prometedor.

Estos experimentos se encuentran en el archivo *perceptron_fourier.ipynb*

5.4. LSTM

Por último vamos a cambiar la arquitectura de nuestra red para usar redes recurrentes, concretamente una red LSTM. Definimos cinco capas de

LSTM para que tenga bastante profundidad. A su vez, reducimos el tamaño de las capas internas a 150, frente a las 1000 usadas en el perceptrón. Aún así, el modelo es bastante grande (y algo lento en entrenar, 70 minutos), por lo que, además, añadimos un *dropout* de 0.2 para evitar que memorize los datos. En su forma más basica, aplicando sólo reducción de picos y normalización, el modelo no consigue aprender. En el siguiente paso añadimos el filtro gaussiano, que hace el modelo mejore.

Finalmente, incluimos la última de las técnicas con las que estamos trabajando y probamos a usar SMOTE para ver como la red LSTM responde frente a las falsas instancias. El resultado es un tanto decepcionante, con una sensibilidad de 0.47 y una especificidad de solo 0.59 y ambas funciones de perdida con valores relativamente elevados y oscilando.

El código de estos experimentos se encuentra en el archivo *lstm_base.ipynb*

5.5. Resumen y conclusiones

Para poder estudiar mejor los resultados de los diversos se presentan en una tabla HTML en el archivo *resultados.html*. La tabla permite filtrar los resultados según las características del modelo y mostrando además las gráficas correspondientes al área bajo la curva y la evolución de las perdidas. Un resumen de estos datos puede encontrarse en la tabla Resumen comparativo de modelos 5.1, de donde podemos extraer algunas conclusiones:

- La reducción de picos y la normalización no son suficientes. Ningún modelo que use solo estas dos técnicas ha conseguido aprender nada.
- Los modelos entrenados usando la funcion *BCEWithLogitsLoss* no parecen funcionar.
- A la hora de aplicar el filtro gaussiano, la opción de trabajar solamente con el resultado y no restarlo a la señal original parece ofrecer mejores resultados.
- SMOTE realmente funciona; todos los modelos entrenados con esta técnica presentan buenos resultados y permiten justificar el coste computacional extra.
- El análisis de frecuencias mejora respecto a los modelos simples pero no sobre los modelos entrenados usando SMOTE. La combinación de Fourier con SMOTE funciona bien.

Modelo	Sens.	Espe.	Score	Epoch
perceptron_sgd_cross	0	1	0.4965	0
perceptron_adam_cross	0	1	0.4965	0
perceptron_adam_bce	0	1	0.4965	0
perceptron_adam_cross_mini	0.8571	0.8571	0.7346	2
perceptron_sgd_cross_diferencia	0.7143	0.998	0.8528	86
perceptron_sgd_cross_solo_filtro	1	1	1	32
perceptron_adam_cross_diferencia	0.8571	0.993	0.9178	36
perceptron_adam_cross_solo_filtro	1	1	1	23
perceptron_adam_bce_diferencia	0	1	0.4965	0
perceptron_adam_bce_solo_filtro	0	1	0.4965	0
perceptron_smote_sgd_cross_diferencia	1	0.995	0.995	20
perceptron_smote_sgd_cross_solo_filtro	1	1	1	8
perceptron_smote_adam_cross_diferencia	1	0.996	0.996	3
perceptron_smote_adam_cross_solo_filtro	1	1	1	3
perceptron_sgd_cross_fourier	0.5714	1	0.7834	75
perceptron_sgd_cross_smote_fourier	1	1	0.999	5
lstm_sgd_cross	0	1	0.4965	0
lstm_sgd_cross_diferencia	0.6079	0.496	0.3047	95
lstm_sgd_cross_smote_diferencia	0.478	0.598	0.2895	82

Tabla 5.1: Resumen comparativo de modelos.

- Las redes LSTM no parecen terminar de aprender correctamente. Aunque sus resultados no son totalmente negativos y los modelos con SMOTE ciertamente son capaces de aprender, no parecen encontrarse a la altura del resto.

5.6. Implementación web

Como paso final, vamos a desarrollar una pequeña aplicación web para dar a conocer el proyecto y permitir usar alguno de los modelos entrenados. Para ello nos serviremos de Flask, un framework minimalista de Python que permite un desarrollo sencillo y muy rápido.

Para presentar un poco mejor los resultados, optamos por incluir algunos gráficos, mostrando la proporción entre estrellas con y sin exoplanetas, así generar de forma dinámica las gráficas de flux y de frecuencia para las

estrellas con exoplanetas. Además, localizaremos la web, de forma que esté también disponible en inglés.

Desplegaremos nuestra web en Heroku, *PaaS* gratuito, mediante un pipeline de despliegue continuo integrado en GitLab. De esta forma, cada *push* realizado en la rama *master* realizará un nuevo despliegue de la web.

Hay dos factores importantes a tener en cuenta cuando desplegamos en Heroku. El primero de ellos es que la plataforma establece un límite máximo de 500 MB para las cuentas gratuitas. Esto nos obliga a definir nuestro entorno con cuidado para no sobrepasar dicho límite como, por ejemplo, usar una versión más pequeña (y antigua) de Pytorch, así como instalar solamente la versión CPU. El segundo problema es un posible *timeout* de la aplicación. Heroku lo fija en 30 segundos y no puede ser aumentado, por lo que si la conexión no es muy buena o el usuario trata de cargar un archivo muy grande, el límite puede superarse. Para tratar de evitar esta situación se ha añadido un mensaje informativo en la propia web.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] 3blue1brown - animated math. https://www.3blue1brown.com/. Accessed: 21/05/2020.

[2] Crisp-dm – a standard methodology to ensure a good outcome. https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome. Accessed: 14/05/2020.

[3] Evolución del precio del gb en discos duros. https://www.xataka.com/componentes/1956-gb-disco-duro-salia-a-9-2-millones-dolares-2019-precio-no-llega-a-dos-cen Accessed: 14/05/2020.

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

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

[6] Light curve of a planet transiting its star. https://exoplanets.nasa.gov/resources/280/light-curve-of-a-planet-transiting-its-star. Accessed: 28/04/2020.

[7] Transformada de fourier. https://es.wikipedia.org/wiki/Transformada_de_Fourier. Accessed: 21/05/2020.

[8] Understanding lstms. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 21/05/2020.

[9] NV. Chawla and WP. Kegelmeyer. SMOTE: synthetic minority oversampling technique. *Journal of Artificial Intelligence*, 16:321–357, June 2002.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] J. D. Hartman and G. Á. Bakos. VARTOOLS: A program for analyzing astronomical time-series data. *Astronomy and Computing*, 17:1–72, October 2016.

[12] Michael Hippke and René Heller. Optimized transit detection algorithm to search for periodic transits of small planets. 623:A39, March 2019.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, December 2014.

[14] György Kovács. An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets. *Applied Soft Computing*, 07 2019.

[15] Marvin Minsky and Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[17] A. Viloria-Lanero and V. Cardeñoso-Payo. Integration of skin segmentation methods using anns. 2006.