**Práctica de Machine Learning**

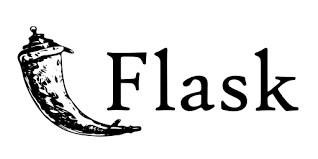
*Productivización del modelo con Flask*

*20 de enero de 2022*

Autores:

*Carlos Sebastiani*

*Guillermo González Díaz*

**Explicación flask:**

En primer lugar hemos creado una carpeta donde hemos incluido:

- **app.py**: una app para poder ejecutar el modelo

- **light\_pipe.ipynb**: un noetbook donde preparamos el modelo para su posterior inclusión en la app. Al final de este script hemos jugado con el umbral de prob\_predictions para ajustarlo manualmente.

- **modelo.sav**: el anterior modelo, ejecutado y guardado para poder cargarlo en la app.

- **productivización**: una subcarpeta con el mismo nombre donde hemos creado un virtual environment para la creación del docker.

Hemos decidido usar el LightGBM, que si bien no es el modelo que mejor resultados nos dio durante la realización del primer trabajo, es el que se ejecutaba más rápidamente. Probamos con el Random Forest, que fue nuestro ganador, pero tras 4 horas de ejecución tuvimos que pararlo, pues el modelo pesa más de 2,7gb.

Para poder ejecutar el LighGBM, tuvimos que realizar algunos cambios, pues al entrenar nuestro modelo, no usábamos un pipe para realizar el preprocesado de las variables, así que lo hemos incluido y re-entrenado el modelo, de esta forma, la aplicación queda mucho más ligera, ya que al cargar el modelo, este incluye dentro el preprocesado de variables para poder luego hacer las predicciones.

La app.py la hemos hecho con Flask. Esta es una librería que permite crear aplicaciones usando Python. La idea detrás de la librería es que usando un microframework se pueda conseguir una aplicación que sea simple de hacer y además pueda ser completa.

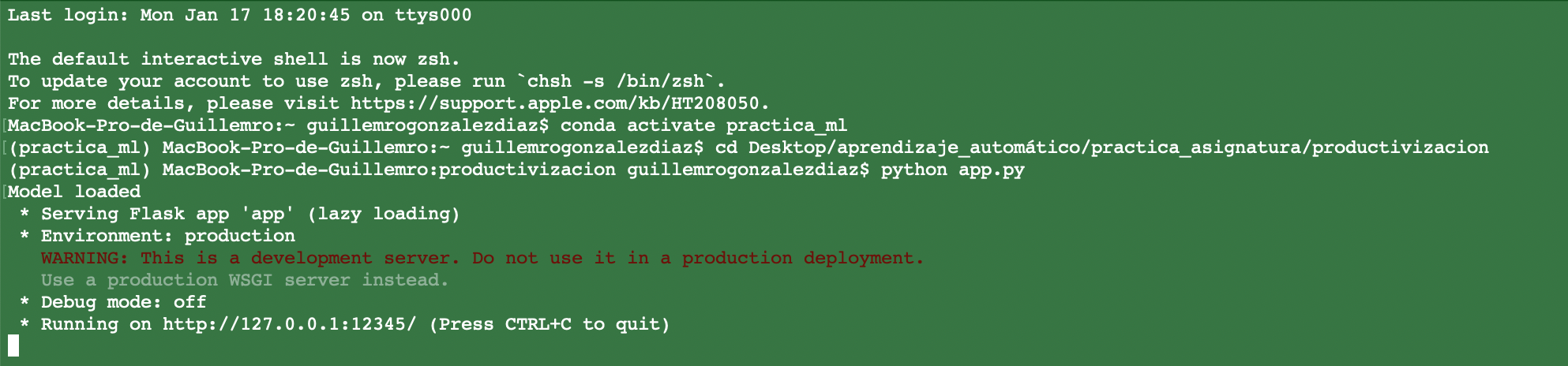
La idea de la aplicación es que podamos introducir valores y esta nos devuelva la predicción, en el caso específico de esta, los valores a introducir serán las variables del accidente (configuración, conductor, clima, etc) y la aplicación nos debería devolver los valores de la predicción, 1 o 0 (fallecimiento o no fallecimiento).

Nuestra aplicación es bastante sencilla, pues se trata de una app en la que hemos definido una función predict, que va a tomar un json con las variables del modelo y predecir. Dentro de la app va realizar unas transformaciones, como pasar el json a un data.frame de pandas, y luego hacer el predica sobre el modelo con el pipe que hemos cargado anteriormente.

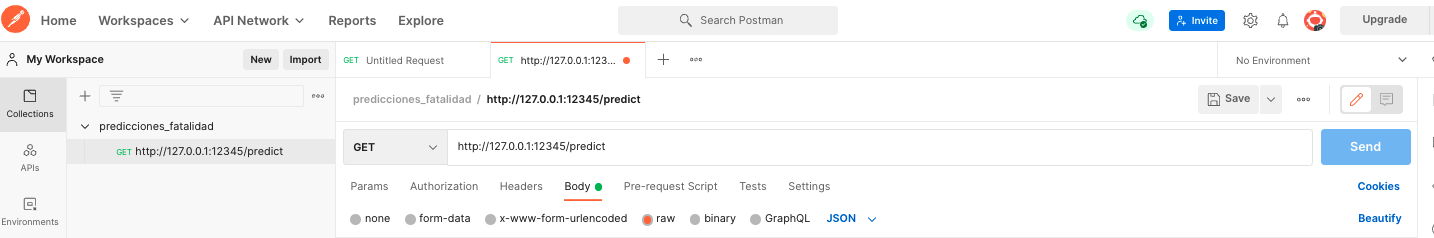
Una vez creada la app, nos hemos registrado en Postman, para poder ejecutar nuestra app. En vista de que la naturaleza de nuestra app es una API, postran será ese UX que usaremos para interactuar con ella.

Postman es una app de Chrome que nos permite construir y testear apis enviando requests.

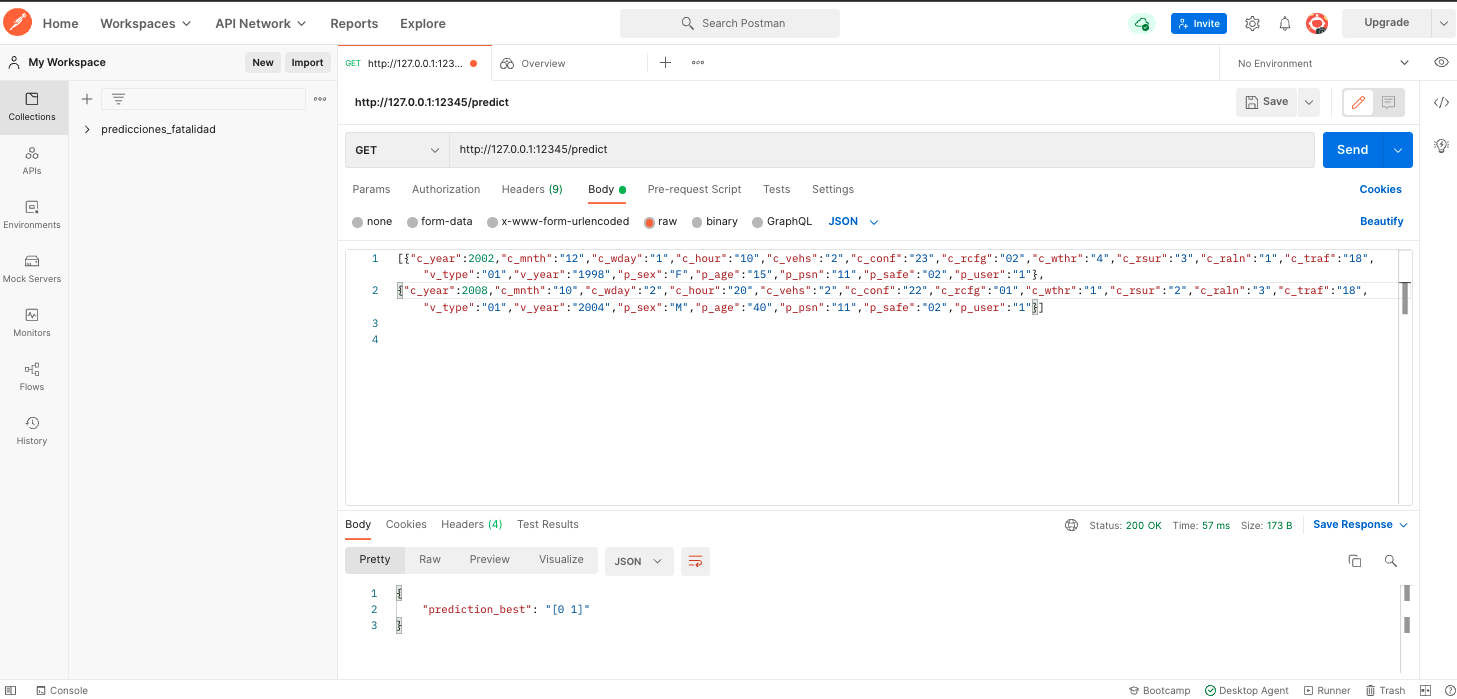
Una vez con este entorno creado, desde la terminal de nuestro ordenador, vamos a nuestra carpeta donde está la app y corremos el comando *python app.py*, y obtenemos lo siguiente:



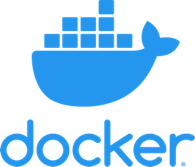
Y la app nos dice que está corriendo en esa http. Entonces ya desde postman con nuestra cuenta creada, introducimos la http que vemos ahí arriba en el buscador, y separado con una barra, el nombre que le hemos puesto a nuestra ruta en la app: *predict* en este caso.



Llegados a este punto, ya tenemos enlazado nuestra app y el modelo con Postman. A continuación, solo teníamos que pasarle las predicciones en formato json. Para comprobar que todo funcionaba correctamente, hemos hecho dos consultas. Hemos extraído dos filas de nuestro dataset de test en los que sabíamos que la primera no estaba involucrada en un accidente mortal, mientras que la segunda sí. Para introducir la request en postman, simplemente hemos pedido ‘GET’, que es lo que habíamos especificado en la app, y en el cuadro de abajo las filas mencionadas anteriormente en formato json. Hay que especificarle que es “raw” y que lo que estamos metiendo es un json. Una vez hecho esto, activamos postman en nuestro escritorio y realizamos la consulta y esto es lo que nos devuelve:



Podemos ver que podemos invocar al modelo, y que este predice correctamente las dos filas mencionadas.

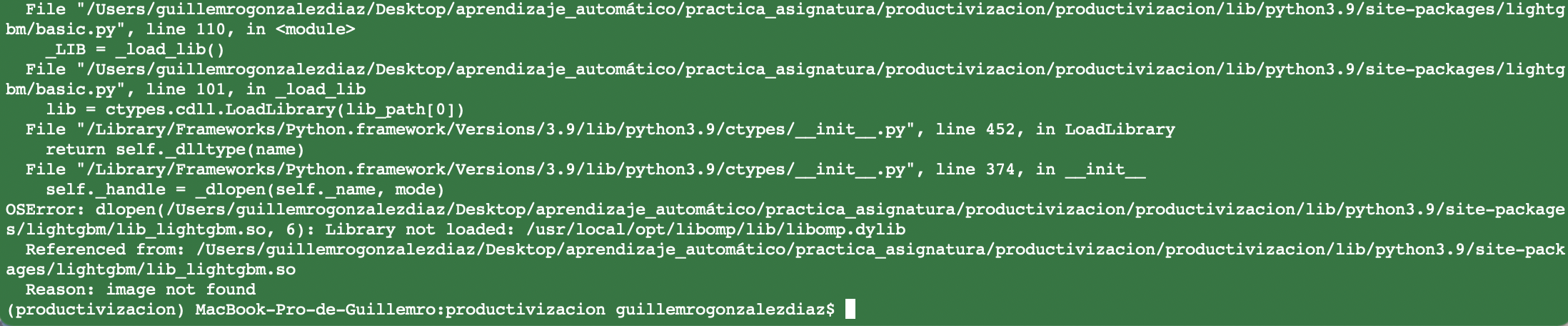


**Explicación Docker:**

En primer lugar, comentar que para nostros ha sido imposible el realizar el docker para poder ejecutar todo, pero lo hemos intentado y aquí están los pasos.

1) Hemos creado un virtual environment desde la carpeta donde teníamos incluido nuestro modelo y nuestra app.py. Es el environment que hemos comentado en el punto 1 y el que está añadido en la entrega de este trabajo.

2) Una vez creado el environment, hemos intentado correr la app. Lógicamente daba error, pues ese environment estaba vacío, y hemos tenido que ir instalando una a una las librerías

que nos hacían falta. Desafortunadamente, hemos tenido problemas al importar el LightGBM y no hemos sido capaces de traerlo a este nuevo virtual environment, este es el error que nos ha parado. Hemos probado con brew install y con varios Lightgmbs de github, como el que está incluido dentro de nuestro environment, pero este error persistente nos ha impedido avanzar, incluso después de haber probado instalando LightGBM desde diversas fuentes.

No obstante, estos son los pasos que habríamos seguido para completar si este modelo no nos hubiera dado problemas:

3) *pip freeze > requirements.txt*. Una vez hecho esto, tendríamos el txt con únicamente las librerías necesarias para poder invocar el modelo.

4) Una vez hecho esto, añadiríamos el *dockerfile*, que es el texto con todos los pasos necesarios para construir la imagen en cualquier docker.

5) Por último, tendríamos que haber corrido este comando *docker build -t cualquier\_nombre*. En este punto ya estaría la imagen creada, y solo habría que correr el contenedor *docker run* y logearnos.