Práctica obligatoria final: Productivización del módelo

Trabajo realizado por:

Pablo Oceja Campanero

Introducción:

En el siguiente documento se va a realizar el resumen y la explicación de los pasos que se han seguido para poder realizar la productivización del modelo realizado a lo largo de las prácticas obligatorias de la asignatura siendo esta práctica y su resultado el paso final para la puesta a punto de un modelo de aprendizaje automático.

1.- Primer paso: descarga del modelo en formato pickle.

Incluido ya en el notebook 4 de la segunda entrega, inserté el código necesario para la descarga del modelo, a continuación se muestra el código empleado:

```
# Save the trained pipeline (including the model and preprocessing steps) to a pickle file
with open('trained_model.pkl', 'wb') as f:
    pickle.dump(pipeline, f)
    print("Model saved as 'trained_model.pkl'")
```

El archivo se guarda en la ruta donde este el propio notebook que lo ejecuta. Los pasos a seguir son:

Primero se importa la librería pickle, que es la que nos va a permitir guardar le modelo en un formato ".pkl", y posteriormente se utilizará dicho archivo para la puesta en producción del modelo. Al final, el archivo pickle/pkl contendrá el modelo final seleccionado en el que estuve trabajando tanto en el notebook 1 como en el notebook 2. En este caso he tenido que realizar un nuevo notebook con los datos del final del notebook2, y aplicarle, a través de pipelines, el OneHotEncoding a las variables categóricas, salvo OCCUPATION_TYPE y ORGANIZATION_TYPE a las cuales se les aplica TargetEncoding, y el escalado de variables a las numéricas. De esta forma obtenemos un pipeline con las siguientes características:

```
# Identify categorical columns
cat_var = X_train.select_dtypes(include='object')

# Identify columns for encoding
onehot_features = [col for col in cat_var.columns if col not in ['OCCUPATION_TYPE', 'ORGANIZATION_TYPE']]
target_encode_features = ['OCCUPATION_TYPE', 'ORGANIZATION_TYPE']

# Preprocessing for OneHotEncoding
onehot_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))  # Generate dense output for OneHotEncoder
])

# Preprocessing for Target Encoding
target_transformer = Pipeline(steps=[
    ('target_encoder', TargetEncoder())
])

# Scaling with StandardScaler for numerical features
scaler = StandardScaler()

# Identify numerical features
numeric_features = X_train.select_dtypes(include=['float64', 'int64']).columns

# Combine the preprocessing steps using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', onehot_transformer, onehot_features),  # Apply OneHotEncoder to other categorical columns
        ('target', target_transformer, target_encode_features),  # Apply TargetEncoder to 'OCCUPATION_TYPE' and 'ORGANIZATION_TYPE'
        ('scaler', scaler, numeric_features)  # Apply StandardScaler only to numerical features
    ],
    remainder='passthrough'  # Other columns are passed through unchanged
)
```

Y me permite en un pipeline tener el preprocesamiento que se ve arriba y aplicar el modelo LGBM junto con el preprocesador en un mismo pipeline:

```
param_dist = {
    'num_leaves': [10],
    'max_depth': [5],
    'learning_rate': [0.1],
    'n_estimators': [100],
    'min_child_samples': [30],
    'min_split_gain': [0.1],
    'subsample': [1.0],
    'colsample_bytree': [0.8],
    'lambda_l1': [0.1],
    'lambda_l2': [0.1]
}

# Initialize the LGBM model with the specified parameters
lgbm_model = LGBMClassifier(
    num_leaves=param_dist['num_leaves'][0],
    max_depth=param_dist['max_depth'][0],
    learning_rate=param_dist['learning_rate'][0],
    n_estimators=param_dist['n_estimators'][0],
    min_child_samples=param_dist['min_child_samples'][0],
    min_split_gain=param_dist['min_split_gain'][0],
    subsample=param_dist['subsample'][0],
    colsample_bytree=param_dist['colsample_bytree'][0],
    lambda_l1=param_dist['lambda_l1'][0],
    lambda_l2=param_dist['lambda_l2'][0]
)

# Define the complete pipeline with LGBMClassifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),  # Apply preprocessing to the features
    ('model', lgbm_model)  # Use the LGBM model with the specified parameters
])
```

Se puede observar como el pipeline aplica primero el preprocesamiento mencionado anteriormente y luego aplica al modelo.

2.- Segundo paso: creación del archivo app.py:

Una vez tengo guardado mi modelo en un archivo pickle, tenemos que crear un archivo que implementa una **API REST** usando Flask para servir a mi modelo de Machine Learning en producción. Es fundamental para la puesta en producción del modelo porque me permite interactuar con el modelo a través de peticiones HTTP, facilitando su integración en aplicaciones externa. Se puede encontrar el archivo app.py dentro de la carpeta src del proyecto.

Además, junto a app.py en la carpeta de src meto el modelo entrenado “model_trained.pkl” para que lo lea de forma fácil en un futuro.

3.- Tercer paso: creación archivo “Dockerfile” sin formato:

Para poder crear la imagen en Dockerfile Desktop necesitamos un archivo Dockerfile sin ningún tipo de formato en el que se explicaran los pasos a seguir a la hora e crear

la imagen de Dockerfile, para esto tuve que crear en un bloc de notas los paso a seguir y luego guardarlo como “(*) todos los archivos” de esta manera queda un archivo Dockerfile sin formato.

4.- Cuarto paso: crear el html base de la web.

Necesitaba un html que sirviera de base para poder poner los valores de las variables y que me devolviera una predicción, 0 o 1, en función de dichos valores. Para esto se le pidió que escogiera las variables con las que se entrenó el modelo y, dependiendo de si son categóricas, binarias, de decisión o numéricas hacer que fueran seleccionables los valores de las variables (esto en el caso de las 3 primeras) o que el usuario tenga que introducir los valores manualmente (solo las numéricas), de esta forma elimino el que los usuarios puedan poner un valor que no existe, escribir mal el trabajo, la organización o estudios del cliente en cuestión etc.

5.- Quinto paso: crear la imagen en Docker

Una vez tenemos todos los archivos necesarios ya podía crear la imagen en “Docker Desktop” en este caso lo primero que hay que hacer es abrir “Docker Desktop” y a través de “Anaconda Prompt”, y tras llegar a la carpeta de destino, creamos la imagen con el siguiente comando:

“docker build -t practica_productivizacion .”

Y el resultado es el siguiente:

```
(base) C:\Users\pablo\OneDrive\Escritorio\Master\AprendizajeAuto\productivization_practice>docker build -t productivization_practice .
[+] Building 23.9s (15/15) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.08kB
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [ 1/10] FROM docker.io/library/python:3.9-slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a7166951cfea
 => => resolve docker.io/library/python:3.9-slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a7166951cfea
 => [internal] load build context
 => => transferring context: 3.10kB
 => CACHED [ 2/10] RUN apt-get update && apt-get install -y --no-install-recommends     libgomp1     && rm -rf /var/lib/apt/lists/*
 => CACHED [ 3/10] WORKDIR /app
 => CACHED [ 4/10] COPY env/requirements.txt /app/
 => CACHED [ 5/10] RUN pip install --no-cache-dir -r requirements.txt
 => [ 6/10] COPY . /app/
 => [ 7/10] RUN ln -sf /usr/bin/python3 /usr/bin/python
 => [ 8/10] RUN chmod +x src/app.py
 => [ 9/10] RUN chmod 755 web/index.html
 => [10/10] RUN chmod 644 models/trained_model.pkl
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:cd8f83fd8f0f9bf2ecaf3b0035030c748e4bef9081bad3805c71217baab94792
 => => exporting config sha256:15238b89e2ccb1893afb99d75612457bd2d9cf0a2bebc9a2225829140b5efcfc
 => => exporting attestation manifest sha256:2dd8441904208cf987df321d2ee2155de2f8f68de82d59062d647b480beb0a82
 => => exporting manifest list sha256:b8a000786370f8a15f86290cdc7da0d9bfc28449a14c0afca48bf4548de0488b
 => => naming to docker.io/library/productivization_practice:latest
 => => unpacking to docker.io/library/productivization_practice:latest

(base) C:\Users\pablo\OneDrive\Escritorio\Master\AprendizajeAuto\productivization_practice>
```

Vemos como va siguiendo los pasos que se le han indicado en el archivo sin formato denominado como Dockerfile.

Tras esto, hemos de correr la página web en local que se creará con nuestra propia IP a través del siguiente código:

“docker run -p 5000:5000 practica_productivizacion”

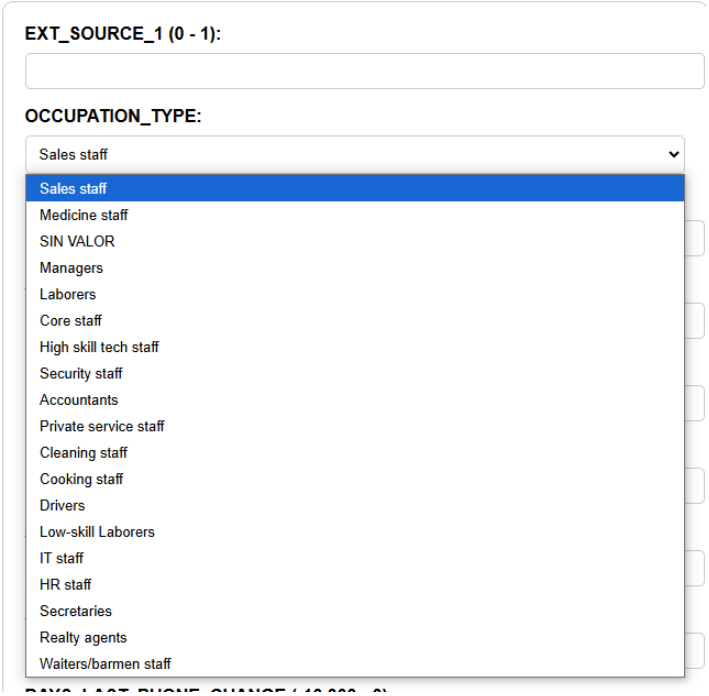
Este comando levanta una aplicación que está empaquetada en la imagen practica_productivizacion y la hace accesible desde el navegador.

Y nos indica que se esta ejecutando correctamente a través del siguiente output:

```
(base) C:\Users\pablo\OneDrive\Escritorio\Master\AprendizajeAuto\practica_productivizacion>docker run -p 5000:5000 practica_productivizacion
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

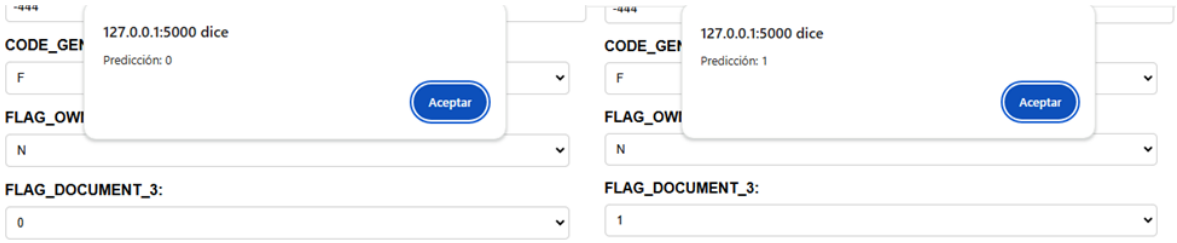
Copiando la primera IP, 127.0.0.1:5000 en el navegador nos lleva a la API que se esta ejecutando y que tiene la siguiente estructura:

Vemos como tiene una estructura muy sencilla y se cumple lo que mencionaba en el paso número 4, que aquellas variables que requieren de un valor. (p.e. EXT_SOURCE_1) te pide que pongas un valor de 0 a 1 y en aquellas categóricas, binarias o de selección (p.e. OCCUPATION_TYPE, CODE_GENDER o FLAG_OWN_CAR) se despliega un menú que luce así:



De esta forma se evitan errores al poner erroneamente un nombre, signos de puntuación…

Además al rellenar todos los valores y darle a enviar nos devuelve si clasifica a dicho cliente como 0 o como 1 como muestra a continuación con unos datos inventados:



6.- Sexto paso: crear las predicciones

Se ha creado un dashboard que ha generado las predicciones que se pueden encontrar en la carpeta de “data” denominado “predictions.csv” y luego a través de plotly se ha realizado alguna visualización de distribución de como se verían dichas predicciones siendo estas visualizaciones un pequeño prototipo de lo que se podría ver como pueden ser distribuciones, rangos… por variable.

Conclusiones

Tras leer y analizar los artículos proporcionados por el docente y añadido a esto la lectura de información extra en internet cuando me encontraba con un fallo, he sido capaz de llevar a cabo la productivización del modelo de forma que escribiendo datos externos a los del dataset original, es decir supuestos clientes nuevos, vemos como es capaz de clasificarlos como 0 o como 1. Pese a que ha sido una practica más corta que las demás ha sido la más difícil en mi opinión ya que no tenía ninguna plantilla de la que partir y ha requerido de mucha investigación y corrección de errores por mi parte y de forma autónoma.