SPRINT 17: ML WRAP-UP

U1) PIPELINES DE SKLEARN

Pipeline:

- encadenar operaciones y aplicarlas a los datos.
- agruparlas
- .predict -> al dataset le hace todas las operaciones que se haría
- facilita los despliegues
- facilita los gridsearch (lo tendréis que ver vosotros)
- dataset del titanic dos versiones:
    - funciones
    - pipelines

Persistencia:

- Pickle
- Joblib: otra librería para guardar objetos en binario
- Normalmente nos dan datos para predecir y no tenemos target. Tal vez después de mucho tiempo nos darán esos datos.
- Normalmente tendremos un “fichero pickle” y un script para que se ejecute. EXPANDIR.

PROYECTO ML SPRINT 17-U1

1. Quitar columnas
2. Imputar nulos
3. Selección de features (mini EDA) … ya hecho
4. Codificar categóricas: PClass, Sex, Embarked (no están Alone, Who)
5. Numéricas: escalado y posible transformación.

Todo lo anterior al train, al test y a los datos nuevos.

Pués vamos a aprender a hacer todo lo anterior con Pipelines.

PIPELINE: Es una secuencia de objetos de scikit-learn

Transformer (fit, transform): StandardScaler, OneHotEncoder, MinMaxScaler, PCA. El transform siempre va a dar igual. No es una estimación.
Estimator:(fit): RandomForestClassifier. Una estimación. El DBScan hace un fit. No hay predict. No es ni un predictor ni un transformer.
Predictor (fit, predict, score): RandomForestRegressor. Todo predictor es un estimator. Es hijo de Estimator.
Un Pipeline hereda lo que sea el último elemento de su cadena.

———————

ColumnTransformer:

- Va a hacer mil cosas.
- Ejemplo ColumnTransformer: ‘nombre_del_Pipe’, acción (drop), lista_de_columnas, remainder = ‘passthrough’.
- La etiqueta sirve para nombrar y para que el GridSearch sepa a dónde envíar parámetros. Sirve para diferenciar dos StandardScaler por ejemplo.

get_features_name_out: devuelve los nombres de las features
- con la etiqueta del nombre_del_transformer
- con __ y después nombre_de_la_variable

Agrupamos funciones para el procesado:
- Por un lado repetimos funciones
- Por otro ya tenemos (Pipeline) de antes

A modelar:
- Regresión Logística, RandomForestClassifier y XGBClassifier
- cross_val_score de train pero con el preprocessing
- ¿cómo sería con pipelines?:
    - Crear un pipeline para el preprocesado y uno para el modelo
    - El pipeline lo trato como un modelo, pero tiene el preprocesado dentro

Ahora un gridsearch:
- Creamos los diccionarios, uno para cada modelo
- Definimos 3 GridSearch y pasamos un grid a a cada GridSearchCV

GridSearch con nombres:
- Para pasar grids a los pipelines, necesitamos el nombre de la etapa del modelo.
- Podriamos pasar también parámetros al SimpleImputer media/mediana

Pipelines II
- Carga del Pipeline junto con el preprocesado
- A diferencia de la funcional, cargar librería, y si hay cambios en la librería

U NLP-Sentiment analysis)

- IMDB reviews dataset.
- Study and model an algorithm to predict if the review is positive or negative based only on the text of the review.
- We train it with labeled reviews (pairs of review and label: positive/negative).

- We train a baseline with BinomialNB
  - It is much faster
  - But considers all variables independent, limitations.
- We train a model based on Deep Neural Networks
  - It still underfits
  - Currently I am running DNN with 37,000,000 params across 4 layers:
    - The RFIDF matrix size weights: 40000 reviews * an alphabet of 97000 words
    - Fitting is very slow (~ hours).

Shape of X_train_vec is (40000 reviews, 92692 tokens) and y_train shape is (40000 reviews,)
We have just over 37,000,000 parameters. If we change the number of perceptrons of the initial layer even more.

In each layer there are less perceptrons therefore they represent a space of lower dimensionality. Such reduction in dimensionality (we start with 92692). In this new lower dimensionality spaces of layers with 800 or 400 perceptrons, the words change their distance (initially, in the 92692 space, they are equally far from one another) and as they start passing through layers they start getting closer, they start forming clusters based on the following criteria: the sentiment of the text. In lower dimensionality spaces they form patterns, hopefully.

Model Summary:
Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 400)	37,077,200
dropout (Dropout)	(None, 400)	0
dense_1 (Dense)	(None, 100)	40,100
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 20)	2,020
dropout_2 (Dropout)	(None, 20)	0
dense_3 (Dense)	(None, 1)	21

Total params: 37,119,341 (141.60 MB)
Trainable params: 37,119,341 (141.60 MB)
Non-trainable params: 0 (0.00 B)

Results obtained with only three layers:

```
Evaluating on the test set...
For the model sentiment_model_dl.pkl, just three Neural Network layers,
s, which the Model will now see for the first time. Overall accuracy:
Accuracy: 0.8517

The Classification Report for the Test-Set

Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.95      0.86      4961
         1.0       0.94      0.75      0.84      5039

    accuracy                           0.85     10000
   macro avg       0.87      0.85      0.85     10000
weighted avg       0.87      0.85      0.85     10000
```

U2) Entornos virtuales

Intro de experiencia como programador y analista. CERN: developement y analysis.

Un entorno virtual es una herramienta que permite crear entornos aislados para proyectos Python. Cada entorno virtual tiene su propio conjunto de paquetes instalados y su propia versión de Python, lo que permite gestionar dependencias de manera más eficiente y evitar conflictos entre proyectos.

Ventajas de usar entornos virtuales

1. Aislamiento de Dependencias: Los entornos virtuales permiten que cada proyecto tenga sus propias dependencias, evitando conflictos con otros proyectos.
2. Reproducibilidad: Facilitan la creación de entornos reproducibles, lo que es útil para compartir proyectos con otros desarrolladores o desplegarlos en producción.
3. Control de Versiones: Puedes especificar versiones específicas de paquetes para garantizar que tu proyecto siempre funcione con una configuración conocida.
4. Facilidad de Gestión: Simplifican la gestión de dependencias y versiones de paquetes.

Crear y usar un entorno virtual en Python

1. Instalar `virtualenv` o usar `venv`

Python 3.3 y versiones posteriores incluyen `venv` en la biblioteca estándar. Para versiones anteriores, puedes instalar `virtualenv`:

Una vez activado, cualquier paquete que instales usando `pip` se instalará en ese entorno

Crear un entorno virtual

Usa el siguiente comando:

python -m venv nombre_del_entorno

Entrar en un entorno virtual

Usa los siguientes comandos:

cd \Scripts

activate.bat

deactivate

Ahora puedes hacer cualquier cosa con los paquetes y pip. Tendrás como una instalación de python personal para ese proyecto.

Desactivar el entorno virtual

Para salir del entorno virtual y volver al entorno global de Python, usa **deactivate**.

Requisitos y dependencias

Para compartir tu proyecto, es una buena práctica crear un archivo `requirements.txt` que liste todas las dependencias del proyecto:

Luego, otros usuarios pueden instalar todas las dependencias usando: usar entornos virtuales es esencial para mantener tus proyectos Python organizados y libres de conflictos de dependencias.

CODIGO

python -m env keeras_deep
cd keras_deep\Scripts
activate.bat
deactivate
pip list

Si scripts desactivados
cmd

ML PROJECT FROM SCRATCH

En grupo.