Contents

1 PyCaret

PyCaret es una de las librerías de Python más populares para el desarrollo de modelos de Machine Learning. Con Pycaret puedes hacer muchas cosas, cómo:

- Aplicar imputación de valores perdidos, escalado, feature engineering o feature selection de una forma muy sencilla, tan solo con tan solo indicar unos parámetros.
- Entrenar más de 100 modelos de machine learning, de todo tipo (clasificación, regresión, forecasting) *con una sola línea de código.
- Registrar los modelos entrenados en MLFlow de una forma muy sencilla.
- Crear una API o un Docker para poner el modelo en producción.
- Subir tu modelo a la nube para poder agilizar el despliegue en producción.

Como ves, PyCaret sirve para muchas cosas. Por eso, considero que es una herramienta que todo Data Scientist debe conocer. ¿Te suena bien? ¡Vamos con ello!

1.1 Instalación

Lo primero de todo para instalar Pycaret debes tener dos cuestiones en cuenta:

1. Pycaret solo funciona con las versiones de Python 3.8 o superior en Ubuntu. Por tanto, si tu versión de Python es superior a la 3.8, o bien bajas de versión de Python o, sino, puedes probar Pycaret en Google Colab o en un Docker. Puedes comprobar tu versión de python con el siguiente comando:

```
python --version
```

2. Pycaret utiiza la versión 0.23.2 de Scikit-Learn. Por tanto, si planeas usar una versión más moderna de Scikit-Learn en el mismo proyecto, te recomendaría tener entornos virtuales separados.

3. Con docker

   (a) Primera ejecución:

```
docker run --name pycaretF /
 -v $PWD:/home/jovyan/work /
 -p 8888:8888 pycaret/full # o pycaret/slim
```

   (b) Segunda y sucesivas ejecuciones

```
docker start pycaretF
docker exec pycaretF  start.sh jupyter notebook
```

- Instalación versión 3.0-rc. La versión estable de pycaret da muchos problemas de instalación por las versiones de scikit-learn y otros paquetes. Podemos instalar la versión candidata de la versión 3.0 con:

```
pip install --pre pycaret
```

  La documentación de esta versión, que es algo diferente, podemos verla aquí: Documentation Latest

2 Cómo entrenar modelos en Pycaret

Pycaret cuenta con varios módulos, cada uno de ellos especializado en diferentes tipos de Machine Learning:

- Modelos Supervisados:
- Regresión o series temporales: pycaret.regression
- Clasificación: pycaret.classification
- Modelos No Supervisados
- Clustering: pycaret.clustering
- Detección de anomalias: pycaret.anomaly
- Reglas de asociación: pycaret.arules
- Topic Modeling: pycaret.nlp

Como ves, Pycaret cuenta con muchos módulos diferentes. Aunque esto pueda parecer complejo, la realidad es que el funcionamiento en todos los casos es siempre el mismo:

2.1 Definición del setup

Es el punto más importante para la realización de predicciones. En este punto se definen cuestiones como:

- Preprocesamiento de datos, incluyendo normalización, estandarización, feature selection, feature engineering, feature generation, etc.
- Estrategia de entrenamiento: indica cuestiones como el tipo de estrategia que se va a aplicar en validación, el número de folds en el que se va a aplicar, el número de CPUs a usar en el entrenamiento, si se usa o no GPU, etc.
- Otras cuestiones, como si se debe hacer un log de los resultados obtenidos o el nombre del experimento.

Todo este proceso se realiza usando la función setup.

2.2 Entrenamiento de los modelos

En este caso hay dos enfoques disponibles:

- Entrenar muchos modelos diferentes y ver qué tal funciona cada uno de ellos. Esto lo puedes realizar con la función compare_models.
- Entrenar un único modelo de nuestra elección. Esto lo puedes realizar con la función create_model, a la cual le deberás pasar uno de los identificadores que aparezca en la función models().

Personalmente el enfoque que suelo realizar es el siguiente: cuando se trata del primer acercamiento a un proyecto, primero entrenar muchos modelos diferentes para ver qué tal se comporta cada uno de ellos.

Una vez tengas identificados los modelos que suelen funcionar mejor, te centras únicamente en dichos modelos.

- **Nota**: En el caso de los modelos de series temporales que se reentrenan de forma automática mediante MLOps, si el tiempo de reentrenamiento no es un problema, suele ser interesante entrenar muchos modelos diferentes.

  Asimismo, la función compare_models tiene una serie de parámetros que ayudan mucho en el entrenamiento, tales como:

  - include: se refiere a los ids de los modelos que quieras entrenar.
  - exclude: permite indicar los ids de los modelos que no quieres que se entrenen.
  - n_select: por defecto compare_models siempre devolverá el modelo que mejor funcione. Sin embargo, si n_select > 1, devolverá una lista con los top n mejores modelos. Esto es interesante si queremos, por ejemplo, realizar un modelo de ensemble.

- budget_time: indica el tiempo máximo que estar ejecutándose una función. Es muy util si necesitas que la ejecución del entrenamiento dure menos de X tiempo.
- parallel: permite realizar el entrenamiento en sistemas distribuidos de Spark o Dask.

Por su parte, la función create_model tiene ciertos parámetros interesantes como los siguientes:

- probability_threshold: permite indicar el límite de probabilidad para pertenencia a una clase (por defecto es 0.5).
- Al permitir kwargs, la función permite pasar el valor de los parámetros que usará la función a entrenar. Por ejemplo,si queremos entrenar un Random Forest Classifier (id = rf) y que siempre utilice el valor max_depth = 3, podríamos hacer lo siguiente:

```
from pycaret.classification import setup, create_model
setup()
create_model('rf', max_depth = 3)
```

2.3 Comprensión y evaluación del modelo

Con el paso anterior ya habríamos seleccionado uno o varios modelos. Sin embargo, seguramente queramos ver cómo funcionan esos modelos (qué variables son más relevantes, curvas de aprendizaje, etc.).

Todo esto podemos hacerlo de una forma muy sencilla en Pycaret gracias a la función `plot_model`.

Más concretamente, con el parámetro plot de la función plot_model podemos obtener diferentes tipos de gráficos:

Model Type	Plot Name	Parameter Value
Regression	Residuals Plot	residuals
Regression	Learning Curve	learning
Regression	Feature Importance	feature_all
Regression	Predictions Error	error
Classification	Confussion Matrix	confusion_matrix
Classification	AUC	auc
Classification	Learning Curve	learning
Classification	Feature Importance	feature
Classification	Decision Boundary	boundary
Clustering	2nd PCA Plot	cluster
Clustering	Elbow Plot	elbow
Clustering	Silhouette Plot	silhouette
Clustering	Distance Plot	distance
Topic Modeling	Interactive Topid	topic_model
Topic Modeling	Análisis de Sentimiento	sentiment
Topic Modeling	Análisis de Bigramas	bigram

- Nota. La función plot_model de PyCaret es muy extensa. En este documento cubriré las funciones más utilizadas/comunes. Si quieres aprender todo lo que ofrece esta función te

recomiendo que leas esta página.

Asimismo, la función evaluate_model abre una ventana donde podrás elegir cada uno de los diferentes tipos de gráficos para así evaluar el modelo de una forma sencilla e interactiva.

Además, la función dashboard nos creará un dashboard interactivo basado en Explainer Dashboard.

Por último, la función deep_check usa la librería deepchecks para comprobar si existe o no algún problema en el proceso de entrenamiento como Data Leakage.

2.4 Guardado del modelo para el despliegue

Una vez hayamos entendido mejor cómo y por qué funciona el modelo, podemos ponerlo en producción.

En este sentido, PyCaret nos ayuda a hacer dos tipos de despliegue:

- Despliegues manuales: consiste en guardar el modelo en local para ponerlo nosotros maualmente en producción. En este proceso lo más típico suele ser crear una API y Dockerizar el servicio. **Google CLoud docker**
- Despliegue en la nube: se trata de una serie de funciones para facilitar la subida del modelo a AWS, Azure y GCP. Actualmente únicamente permite hacer la subida al Data Lake (S3, Cloud Storage, Azure Storage), el despliegue posterior del DataLake al servicio quedaría a cargo de los ML Engineers.

2.4.1 Funciones para el despliegue manual

En este proceso lo primero de todo será guardar el modelo.

Para guardar un modelo entrenado con PyCaret podemos usar la funcion save_model, la cual guarda tanto el modelo entrenado como el pipeline de preprocesamiento de datos como un fichero .pickle.

Por otro lado, para poder cargar el modelo previamente guardado se puede utilizar la función load_model.

Asimismo, en caso de que vayamos a hacer la puesta en producción diferente a Python (C, Java, Go, C#), podemos convertir el proceso de decisión de nuestro modelo a dichos lenguajes usando la función convert_model.

Sin emabrgo, si vamos a usar Python para crear una API y después ponerlo en producción, PyCaret incluye la funcion create_api, la cual crear una API basada en FastAPI donde se expone nuestro modelo a peticiones POST.

Nota: APIs con Flask y FastAPI.

Asimismo, si quieres Dockerizar la API, PyCaret cuenta con la función create_docker, la cual te permite Dockerizar la API previamente creada. En este caso, PyCaret creará tanto el fichero Dockerfile como el fichero requirements.txt. **Nota: Docker en más profundidad**

Ahora que conocemos todos los ingredientes que componen PyCaret, veamos cómo usarlo en cuatro casos diferentes: modelo de regresión, de clasificación, de series temporales y de clustering. ¡Vamos con ello!

3 Cómo crear un modelo de regresión con PyCaret

3.1 Carga de los datos

Para crear el modelo de regresión vamos a usar los datos de precios de las casas California, el cual se encuentra en el módulo datasets de Sklearn.

En resumen, se trata de un dataset real en el que tenemos información sobre la casa (Nº de dormitorios, años de la casa, ubicación) y del lugar (población y renta) y con ello deberemos predecir el precio de las casas.

Veamos cómo es el dataset:

```
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing(as_frame=True)
california_housing.frame.head()
```

3.2 Definición del Setup

Perfecto, ahora que tenemos el dataset, vamos a importar las librerías necesarias de PyCaret. Lo más normal suele ser importar todas las funciones, aunque para mayor explicabilidad importaré cada una de las funciones por separado:

```
from pycaret.regression import setup, compare_models, create_model, tune_model, \
  plot_model, save_model

california_housing_setup = setup(
    data = california_housing.frame,
    target = 'MedHouseVal',
    normalize = True,
    transformation = True,
    remove_multicollinearity = True,
    multicollinearity_threshold = 0.8,
    feature_selection = True,
    ignore_low_variance = True,
    remove_outliers = True,
    imputation_type = 'simple',
    numeric_imputation = 'median',
    silent = True
)
```

3.3 Comparación de modelos

Ahora que hemos definido el setup, vamos a seguir la siguiente estrategia:

- Entrenar muchos modelos para ver cuáles funcionan mejor. 2
- Sabiendo cuál es el modelo que mejor funciona, hacer un tuning específico de ese modelo.

Vamos con el primero paso: vamos a comparar muchos modelos:

```
best_model = compare_models()
```

Como podemos ver, se han entrenado 18 modelos diferentes y en cada uno de ellos se ha realizado KFold 10 veces.

Además, el modelo Light Gradient Boosting Machine es el que mejor funciona (MAE = 0.3905) y funciona significativamente mejor que un modelo muy simple como es Dummy Regressor (MAE = 0.8989).

3.4 Tuning de los hiperparámetros

Ahora que sabemos esto, vamos a tunear nuestro modelo de una forma más precisa. Y es que el tuning que realiza PyCaret al entrenar varios modelos suele ser bastante mejorable:

Como puedes ver, gracias a la función tune_model podemos hacer un tuning del modelo mucho más preciso consiguiendo así un modelo mejor tuneado.

De hecho, el modelo ha pasado de tener un MAE en test de 0.3905 a tener un MAE de 0.3809, lo cual es una ganancia considerable. Y eso tan solo aplicando una función y de una forma bastante sencilla.

3.5 Entendimiento y validación del modelo

Por último, vamos a ver cómo se está comportando nuestro modelo. Además, usaremos deepchecks para comprobar que no haya habido ningun problema de data leakage.

Lo primero de todo vamos a comprobar el proceso de aprendizaje con un plot de learning.

```
plot_model(best_model, plot = 'learning')
```

Como vemos, a medida que el modelo se ha ido entrenando ha ido mejorando su capacidad predictiva sobre datos nuevos (cross validation) a medida que disminuía su capacidad sobre train. Por tanto, parece que el modelo es capaz de generalizar de forma correcta.

Ahora, veamos a ver cómo se comportan los residuos en train y test:

```
plot_model(best_model, plot = 'residuals')
```

Como podemos ver,parece que los residuos tanto en train como test siguen una distribución normal. Además, en ambos casos el R2 de los residuos son bastante elevados, por lo que parece que el modelo está bien ajustado.

Así pues, por último vamos a guardar el modelo para ponerlo en produccion.

4 Crear una API Dockerizada con PyCaret

Para poner el modelo en producción voy a exponer el modelo en una APIy lugeo lo voy a Dockerizar para poder desplegarlo en Kubernetes (por ejemplo).

Para crear la API usaré la función create_api y posteriamente para crear el docker usaré la función create_docker.

```
from pycaret.classification import create_api, create_docker

create_api(best_model, 'lightgbm')
create_docker('lightgbm')
```

Si analizamos la carpeta veremos que se habrán creado dos nuevos ficheros: Dockerfile y requirements.txt. Veamos qué tiene cada uno de ellos:

```
# requirements.txt
pycaret
fastapi
uvicorn
```

```
# Dockerfile
FROM python:3.8-slim
WORKDIR /app
ADD . /app
RUN apt-get update && apt-get install -y libgomp1
RUN pip install -r requirements.txt
EXPOSE 8000
CMD ["python", "lightgbm.py"]
```

Con esto ya tendríamos nuestro modelo de regresión listo para ponerlo en producción. Como ves, con PyCaret hemos podido realizar un proyecto de Machine Learning de una forma muy sencilla y rápida.

Ahora, veamos cómo usar PyCaret para un proyecto de Clasificación. ¡Vamos con ello!

5 Cómo crear un modelo de clasificación con PyCaret

5.1 Carga de Datos

Para ver cómo funciona PyCaret en un proyecto de clasificación de texto, lo primero de todo debemos contar con unos datos sobre los que entrenar el modelo.

En este caso, vamosa trabajar sobre el dataset Breast Cancer, el cual incluye distintas métricas de diferentes cáncerees de mama e indica si el cancer es benigno o maligno. Este dataset está disponible dentro de

```
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer(as_frame=True)
breast_cancer.frame.head()
```

Como podemos ver en la siguiente imagen, este dataset cuenta con muchas variables con una alta correlación entre ellas. Al fin y al cabo, ocurre dos cosas:

- Métricas que son resultados de transformaciones de otras métricas.

- Para cada métrica se incluye la media, desviación típica y peor valor obtenido. Generalmente, estas tres métricas suelen estar correlacionadas.

Sabiendo esto de nuestros datos, veamos si Pycaret es capaz de detectar la correlación y quedarse con aquellas variables que tienen menos correlación.

5.2 Definición del setup para clasificación

Al tratase de un problema de clasificación, las funciones principales de PyCaret, tales como setup, compare_models, tune_model, etc. se encuentran dentro del módulo classification.

Por otro lado, como contamos con problemas de multiconlinealidad y correlación, vamos a indicar a a PyCaret que se encargue de solucionar estos problemas.

Para ello vamos a hacer lo siguiente:

- Para eliminar la correlación vamos a fijar el parámetro `remove_multicollinearity` en `True`. Además, con el parámetro `multicollinearity_threshold` podemos indicar a partir de qué nivel de multicolinearidad se eliminar los valores. Por defecto el valor está fijado en 0.9.
- Para eliminar la multicolinearidad, vamos a aplicar PCA. Para ello, indicaremos el parámetro `pca = True`. Asimismo, debemos indicar o el porcentaje de varianza tenemos que ser capaces de explicar o el número de variables con el que nos queremos quedar. Esto lo podemos hacer con el parámetro `pca_components`.
- **Nota**: Aplicar PCA para eliminar la multicolinealidad no es posible en aquellos casos que necesitemos poder interpretar el modelo. En estos casos tendremos que eliminar las variables de forma manual aplicando el *Variance Inflation Factor* o *VIF*.

  Así pues, vamos a crear el setup para nuestro proyecto de clasificación de cancer de mama con Pycaret:

```
from pycaret.classification import setup, compare_models, tune_model, plot_mo
  save_model, load_model

california_housing_setup = setup(
    data = breast_cancer.frame,
    target = 'target',
    normalize = True,
    transformation = True,
    pca = True,
    pca_components = 0.8,
    remove_multicollinearity = True,
    multicollinearity_threshold = 0.8,
    ignore_low_variance = True,
    remove_outliers = True,
    imputation_type = 'simple',
    numeric_imputation = 'median',
    silent = True
```

```
)
```

Por último, vamos a ver cómo quedan nuestros datos transformados:

Como podemos ver hemos reducido el dataset y nos hemos quedado únicamente con 5 componentes principales. Ahora veamos a ver qué tal funciona el entrenamiento de los modelos.

5.3 Entrenamiento de modelos de clasificación con PyCaret

Al igual que en el caso de los problemas de regresión, podemos usar la función compare_models para entrenar muchos modelos diferentes.

En concreto, PyCarete entrena 14 modelso diferentes, desde modelos lineales (Regresión Logística, Linear Discriminant, Ridge) a modelos basados en árboles de decisión (Arbol de decisión, Random Forest, AdaBoost, Gradient Boosting, etc.) y otro tipo de modeos como SMVs, KNN o Naive Bayes.

Así pues, vamos a quedarnos con los 3 modelos de PyCaret que mejor funcionan:

```
models = compare_models(sort = 'AUC', n_select = 3)
```

Si analizamos el objeto `models`, veremos que se trata de una lista que cuenta con 3 valores diferentes, cada uno de ellos con el modelo entrenado:

```
models

[LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=1000,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=7632, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0,
                               store_covariance=False, tol=0.0001),
 LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                            solver='svd', store_covariance=False, tol=0.0001)]
```

Esta vez como los modelos que mejor funcionan son modelos relativamente simples, no vamos a tunear más dichos modelos. En su lugar, vamos a detenernos en evaluar el funcionamiento de dichos modelos.

5.4 Evaluación del rendimiento de los modelos

Tal como he indicado en la parte teórica, una de las claves de PyCaret es que facilita mucho el entrenamiento, pero también la evaluación de los modelos.

En este sentido una de las evaluaciones interesantes es la de deepchecks,la cual genera un report incluyendo varios gráficos de areas que el modelo aprueba o no:

```
pip install pycaret[analysis]
from pycaret.classification import deep_check
deep_check(models[0])
```

Como podemos, el modelo se encuentra perfectamente calibrado, más allá de que en algún segmento la capacidad predictiva del modelo sea del 75%.

Con esto ya podríamos guardar el modelo para hacer predicciones, tal como lo hemos realizado en el caso de la regresión.

6 Cómo y cuándo usar PyCaret

En mi opinión PyCaret es una librería para generación de modelos de Machine Learning low code muy interesante por varios cuestiones:

- Permite entrenar de forma muy sencilla diferentes modelos, por lo que tardas menos que si lo hicieras manualmente en Sklearn. Estos modelos sirven tanto para regresión como para clasificación.
- Permite realizar preprocesamiento de datos de una forma sencilla mediante indicando diferentes parámetros en la función setup.
- Incluye muchas funciones y librerías para poder interpretar los modelos de una forma muy sencilla.
- Te ayuda a crear la API y Dockerfiles de cara a la puesta en producción.

Aunque PyCaret sea muy potente, en mi opinión no es ideal en todas las circunstancias.

Personalmente, PyCaret me parece muy interesante como primer acercamiento en un proyecto de Ciencia de Datos que requiera de Machine Learning.

El motivo es sencillo: de una forma sencilla PyCaret permite entrenar muchos modelos diferentes, por lo que, de una forma sencilla, puedes saber qué modelos son los que posiblemente mejor funcionen.

Sin embargo, de cara a conseguir afinar y mejorar la capacidad predictiva del modelo, suelo optar por realizar optimizaciones manuales de los hiperparámetros, así como otras opciones como modelos de stacking (si el proyecto lo permite).

Aunque PyCaret pueda permitir realizar la optimización mediante otras librerías como SkOptimize, no aporta muchas ventajas en ese sentido. Además, tampoco está pensado para entrenar modelos de stacking.

Además, la versión actual de PyCaret es muy inflexible en ciertas cuestiones, tal como la creación del fichero logs.log, el cual se genera en la carpeta raiz del proyecto. Esto añade ciertos pasos a utilizar PyCaret en procesos de MLOps en ciertas herramientas como Cloud Functions.

En cualquier caso, PyCaret es una grandísima herramienta que, personalmente, considero que todo Data Scientist o Machine Learning Engineer debería conocer.

Espero que este post te haya servido para conocer PyCaret más en profundidad y que te sea util en tu día a día a la hora de crear modelos de Machine Learning.

https://anderfernandez.com/blog/pycaret-low-code-machine-learning-en-python/