Contents

1 PyCaret **1**

2 Instalar PyCaret **2**
- 2.1 Con «conda» . . . 2
- 2.2 Con pip . . . 2
- 2.3 Version pre-release (con soporte para python 3.10, se supone) . . . 2
- 2.4 Desde el código fuente . . . 2
- 2.5 Con docker . . . 2

3 El Módulo de Clasificación de PyCaret **3**
- 3.1 Cargando los datos —Dataset Iris— . . . 3
  - 3.1.1 Análisis Exploratorio de Datos —Exploratory Data Analysis (EDA)— . . . 4
  - 3.1.2 3.3 Inicializar un Entorno (environment) PyCaret . . . 6
  - 3.1.3 3.4 Ver los datos preprocesados . . . 6
  - 3.1.4 3.5 Comparando diferentes modelos . . . 6
  - 3.1.5 3.6 Creación de un modelo con PyCaret . . . 7
  - 3.1.6 3.7 Ajustar un modelo . . . 7
  - 3.1.7 3.8 Trazar/dibujar el rendimiento del modelo . . . 7
  - 3.1.8 3.9 Hacer predicciones sobre nuevos datos . . . 7
  - 3.1.9 3.10 Interpretación del modelo (SHAP module) . . . 7
  - 3.1.10 3.11 Despliegue(deploy) del modelo . . . 7

1 PyCaret

PyCaret es una biblioteca de aprendizaje automático de código abierto y de bajo código en Python que automatiza el proceso de aprendizaje automático. Está diseñado para hacer que el proceso de construcción, entrenamiento y despliegue de modelos de aprendizaje automático sea más fácil y más rápido. Con PyCaret, puede construir, entrenar y desplegar rápidamente modelos de aprendizaje automático con sólo unas pocas líneas de código. Proporciona una completa suite de funciones para preprocesamiento, ingeniería de características, selección de modelos y evaluación de modelos. PyCaret también incluye herramientas para ajustar automáticamente los hiperparámetros, crear modelos en ensamblado y el despliegue de modelos en producción.

PyCaret es un envoltorio de Python sobre otras librerías tales como scikit-learn, XGBoost, LightGBM, CatBoost, spaCy, Optuna, Hyperopt, Ray y algunas más.

El diseño y la simplicidad de PyCaret están inspirados en el papel emergente de los científicos de datos ciudadanos, un término utilizado por primera vez por Gartner. Los científicos de datos ciudadanos son usuarios avanzados que pueden realizar tareas analíticas simples y moderadamente sofisticadas que anteriormente habrían requerido más experiencia técnica.

La mayoría de los profesionales del aprendizaje automático comienzan a experimentar con la establecida biblioteca scikit-learn, pero existe una alternativa más sencilla y accesible: PyCaret.

Esta biblioteca tiene muchas ventajas en comparación con scikit-learn, especialmente para personas con experiencia limitada. Ahora veremos una descripción general de las características prin-

cipales de PyCaret, así como un estudio de un caso centrado en la regresión. Sugiero instalar la última versión de Anaconda en Windows/macOS/Linux para seguir este tutorial, pero también es compatible con Google Colab. Puedes ejecutar el código en un cuaderno Jupyter o usar tu IDE preferido.

2 Instalar PyCaret

2.1 Con «conda»

```
conda install -c conda-forge pycaret
```

2.2 Con pip

```
# create a conda environment
conda create --name yourenvname python=3.8

# activate conda environment
conda activate yourenvname

# install pycaret
pip install pycaret

# create notebook kernel
python -m ipykernel install --user --name yourenvname --display-name "display-nam
```

2.3 Version pre-release (con soporte para python 3.10, se supone)

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

2.4 Desde el código fuente

```
pip install git+https://github.com/pycaret/pycaret.git#egg=pycaret
#ejecutar test
pytest pycaret
```

2.5 Con docker

Docker usa contenedores para crear entornos virtuales que aíslan una instalación de PyCaret del resto del sistema. El contenedor docker de PyCaret viene con un entorno Notebook preinstalado, que puede compartir recursos con su máquina host (acceder a directorios, usar la GPU, conectarse a Internet, etc.). Las imágenes de PyCaret Docker se prueban para cada versión.

```
docker run -p 8888:8888 pycaret/slim
```

Para una imagen con la versión completa de la imagen docker:

```
docker run -p 8888:8888 pycaret/full
```

Ejecutar docker con un volumen externo, por ejemplo donde vayamos a tener el proyecto, en este caso he puesto el directorio actual de trabajo:

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

3 El Módulo de Clasificación de PyCaret

En este tutorial, aprenderemos a utilizar el módulo de clasificación de PyCaret para crear modelos de clasificación a partir de datos.

La clasificación es una tarea básica de aprendizaje automático supervisado que asigna una etiqueta a cada observación de un conjunto de datos. Las etiquetas pueden ser binarias, categóricas o ordinales.

El módulo de clasificación de PyCaret, que usa sklearn bajo el capó, le permite crear y probar modelos de clasificación con unas pocas líneas de código. Incluye una variedad de algoritmos, así como la capacidad de trazar y ajustar hiperparámetros.

Vamos a examinar un caso de estudio de clasificación basado en el módulo de clasificación de PyCaret.

3.1 Cargando los datos —Dataset Iris—

Utilizaremos el conjunto de datos de Iris, que contiene información sobre las características de tres especies de iris: setosa, versicolor y virginica.

El conjunto de datos de Iris es un conjunto de datos de aprendizaje automático supervisado clásico que se utiliza a menudo para probar nuevos algoritmos de aprendizaje automático. El conjunto de datos contiene 150 observaciones de flores de iris, cada una con cuatro características:

- **Longitud del sépalo:** La longitud del sépalo es la longitud de la parte verde inferior de la flor.
- **Ancho del sépalo:** El ancho del sépalo es el ancho de la parte verde inferior de la flor.
- **Longitud del pétalo:** La longitud del pétalo es la longitud de la parte colorida de la flor.
- **Ancho del pétalo:** El ancho del pétalo es el ancho de la parte colorida de la flor.

Las cuatro características se utilizan para predecir la especie de iris de la flor. Hay tres especies de iris: setosa, versicolor y virginica.

El conjunto de datos de Iris es un conjunto de datos de aprendizaje automático muy útil para probar nuevos algoritmos. Es relativamente pequeño y fácil de entender, pero aún así es lo suficientemente complejo como para que sea un desafío para los algoritmos de aprendizaje automático.

Aquí hay un resumen de las características del conjunto de datos de Iris:

- **Número de observaciones:** 150
- **Número de características:** 4
- **Tipo de datos de las características:** 4 numéricas
- **Número de clases:** 3

- **Distribución de las clases:** 50 setosas, 50 versicolores y 50 virginicas

El primer paso es cargar los datos. Podemos hacerlo utilizando la función read_csv() de la biblioteca pandas:

```
from pycaret.datasets import get_data
# Cargar los datos
data = get_data('iris')
```

```
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

Ejemplo de datos:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
data.info()
```

3.1.1 Análisis Exploratorio de Datos —Exploratory Data Analysis (EDA)—

Después de cargar el conjunto de datos, normalmente necesitaremos examinar y comprender sus propiedades básicas, y antes de crear nuestro modelo, es importante realizar un análisis exploratorio de datos (EDA) para comprender los datos y identificar posibles problemas.

En este caso, podemos realizar las siguientes EDA:

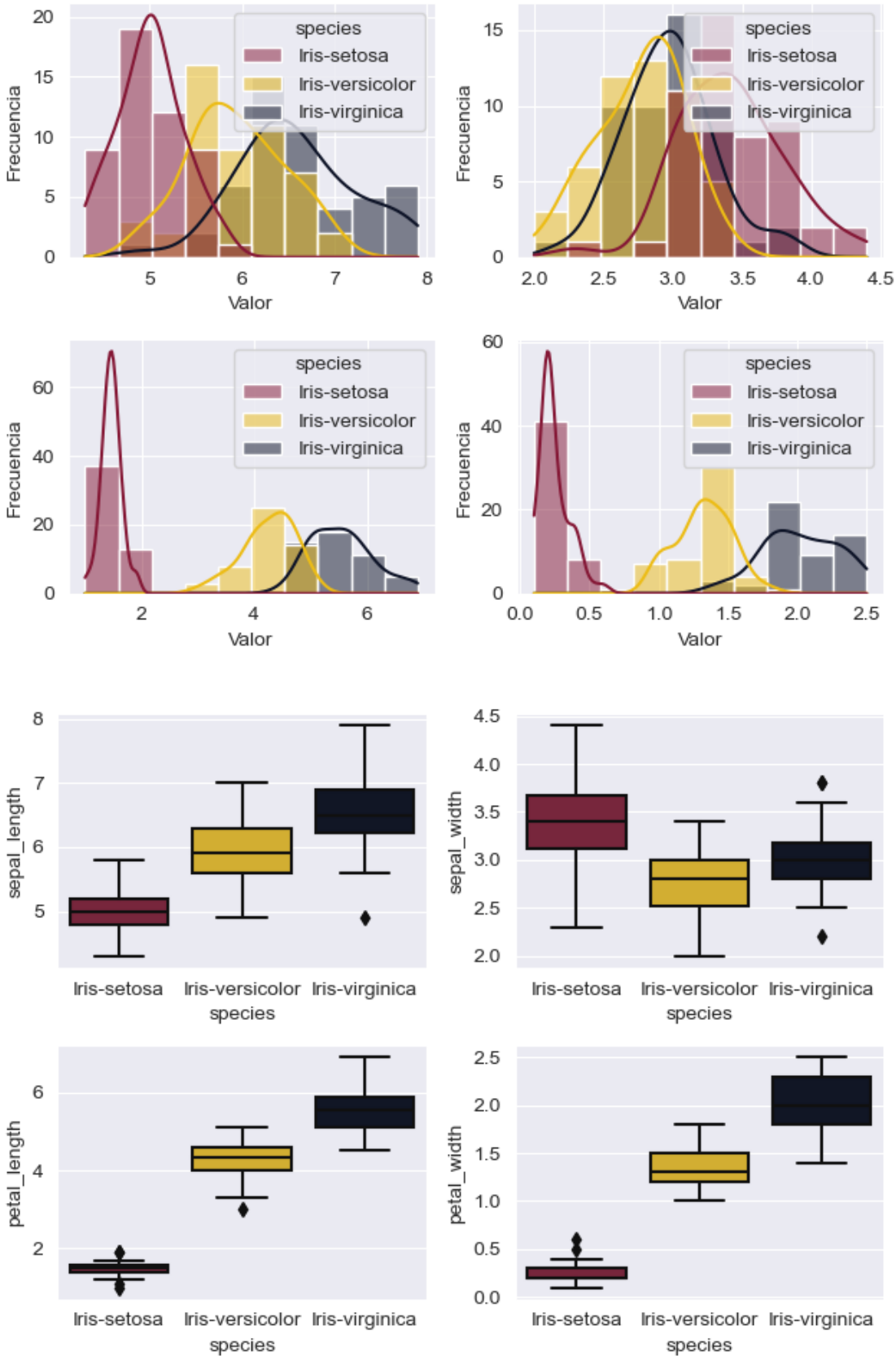
- **Ver los datos**

```
print(data.head())
```

- **Describir los datos**

```
print(data.describe())
```

- **Ver la distribución de las características**


species
Iris-setosa
Iris-versicolor
Iris-virginica
Frecuencia
Valor
sepal_length
sepal_width
petal_length
petal_width
species

- **Comparar las características entre las clases**

```
import pandas as pd
from pycaret.datasets import get_data

import pycaret

# Load the Iris dataset
data = get_data('iris')

# Calculate the mean of each column for each species
for col in data.columns:
    pivot_table = data.pivot_table(values=col, index="species", aggfunc=pd.DataFr
    print(pivot_table)
```

```
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

3.1.2 3.3 Inicializar un Entorno (environment) PyCaret

La función `setup()` de PyCaret inicializa el entorno y prepara la implementación y los datos de modelado de aprendizaje automático. Hay dos parámetros necesarios, un conjunto de datos y la variable objetivo. Después de ejecutar la función, se infiere el tipo de cada característica y se realizan varias tareas de preprocesamiento en los datos.

```
from pycaret.classification import setup

# Inicializar el entorno
clf = setup(data, target = 'species', train_size = 0.8, session_id = 123, normali
```

3.1.3 3.4 Ver los datos preprocesados

PyCaret realiza un preprocesamiento de los datos de forma automática. Podemos ver los datos preprocesados utilizando el siguiente código:

```
print(clf.data.head())
```

```
get_config('X')
```

3.1.4 3.5 Comparando diferentes modelos

PyCaret proporciona una serie de modelos de clasificación que podemos utilizar. Podemos comparar estos modelos utilizando el siguiente código:

```
models = clf.compare_models()

# Imprimir los resultados de la comparación
```

```
print(models)
```

3.1.5 3.6 Creación de un modelo con PyCaret

Una vez que hayamos comparado los diferentes modelos, podemos crear un modelo específico utilizando el siguiente código:

```
# Crear un modelo de Random Forest
rf = clf.create_model("rf")
```

3.1.6 3.7 Ajustar un modelo

Antes de poder utilizar nuestro modelo para hacer predicciones, debemos ajustarlo a los datos de entrenamiento. Esto se realiza utilizando el siguiente código:

```
# Ajustar el modelo
rf.fit()
```

3.1.7 3.8 Trazar/dibujar el rendimiento del modelo

Podemos trazar el rendimiento de nuestro modelo utilizando el siguiente código:

```
# Trazar el rendimiento del modelo
rf.plot_model()
```

3.1.8 3.9 Hacer predicciones sobre nuevos datos

Una vez que nuestro modelo esté ajustado, podemos utilizarlo para hacer predicciones sobre nuevos datos. Esto se realiza utilizando el siguiente código:

```
# Hacer predicciones sobre nuevos datos
predictions = rf.predict(data.drop("Survived", axis=1))
```

3.1.9 3.10 Interpretación del modelo (SHAP module)

Podemos interpretar nuestro modelo utilizando el módulo SHAP. Esto nos ayudará a comprender las características que son más importantes para las predicciones del modelo.

```
# Importar el módulo SHAP
from pycaret.explainers import shap

# Obtener las explicaciones del modelo
shap_values = shap.explain_model(rf, data.drop("Survived", axis=1))
```

Podemos visualizar las explicaciones del modelo utilizando el siguiente código:

```
# Visualizar las explicaciones del modelo
shap.plots.bar(shap_values)
```

3.1.10 3.11 Despliegue(deploy) del modelo

Una vez que nuestro modelo esté listo, podemos desplegarlo para utilizarlo en producción. Esto se puede hacer utilizando una variedad de métodos, como Docker, Kubernetes o Flask.

En este tutorial, hemos aprendido a utilizar el módulo de clasificación de PyCaret para crear modelos de clasificación a partir de datos.