Contents

1 Estructura básica de un modelo con PyCaret 1

2 Tarea 1 con PyCaret: Regresión 1
- 2.1 Datos y objetivos . . . 2
- 2.2 Preguntas sobre el modelo: . . . 2
- 2.3 Ejemplo . . . 3
  - 2.3.1 Más ejemplos . . . 4

1 Estructura básica de un modelo con PyCaret

La estructura básica de un proyecto con PyCaret es la del código fuente a continuación:

```
# importar la libreria
from pycaret.[regression/classification] import *

# cargar los datos
data = pd.read_csv("data.csv")

# iniciar una sesión de pycaret
reg = setup(data, target = 'target_column')

# comparar todos los modelos disponibles
compare_models()

# seleccionar el mejor modelo
best = compare_models()[0]

# entrenar el modelo seleccionado
best_model = create_model(best)

# evaluar el modelo en los datos de prueba
evaluate_model(best_model)

# desplegar el modelo
deploy_model(best_model, model_name = 'best_model')
```

2 Tarea 1 con PyCaret: Regresión

- En primer lugar, vamos a usar el dataset adjunto a la tarea **Airlines Departure Delay 1M**, adjunto a la tarea en Moodle.
- **[Opcional, setup los puede dividir por nosotros]** Una vez que hayan descargado los datos, deben cargarlos en un DataFrame de pandas y dividirlos en conjunto de entrenamiento y prueba utilizando la función train_test_split de sklearn.

- Utilizando la función `setup()` de PyCaret, deben iniciar una sesión de PyCaret y especificar la columna objetivo.
- Utilizando la función `compare_models()`, deben comparar todos los modelos de regresión disponibles y seleccionar el mejor modelo.
- Una vez que hayan seleccionado el mejor modelo, deben entrenarlo utilizando la función `create_model()`.
- Utilice la función `tune_model()` para mejorar el rendimiento del modelo seleccionado mediante la búsqueda de mejores hiperparámetros.
- Utilizando la función `evaluate_model()`, deben evaluar el rendimiento del modelo en el conjunto de prueba.
- Utilice la función `predict_model()` para hacer predicciones en un conjunto de datos de prueba.
- Utilizando la función `plot_model()`, deben generar un gráfico para visualizar los resultados del modelo y hacer una interpretación.
- **Opcional** Finalmente, deben desplegar el modelo utilizando la función `deploy_model()`.

2.1 Datos y objetivos

Los datoas a usar son el dataset "Airlines_DepDelay_1M.csv", y nuestro objetivo es predecir el tiempo de retraso en la salida de los vuelos.

2.2 Preguntas sobre el modelo:

- ¿Cuál es el problema de regresión que están tratando de resolver?
- ¿Qué columna de los datos de retrasos de aerolíneas especificaron como la columna objetivo?
- ¿Qué modelo de regresión seleccionaron como el mejor y por qué?
- ¿Cómo evaluaron el rendimiento del modelo en el conjunto de pruebas? ¿Cuáles fueron las métricas de rendimiento que utilizaron para evaluar el modelo? ¿Cuáles fueron los resultados?
- ¿Cómo visualizaron los resultados del modelo? ¿Qué información pueden extraer de la visualización?
- ¿Cuáles fueron los hiperparámetros que utilizaron para mejorar el rendimiento del modelo?
- ¿Cuáles fueron las características más importantes del modelo?
- ¿Cómo desplegaron el modelo y cómo podría ser utilizado en una aplicación real?

Es importante consultar la documentación oficial

2.3 Ejemplo

El conjunto de datos Diamonds es una colección de datos sobre 150 diamantes vendidos en una subasta [1]. Contiene las siguientes variables: quilates, corte, color, claridad, profundidad, mesa, precio, x, y, z. El quilate es una medida de peso, el corte es una medida de la calidad del corte, el color es una medida del color del diamante, la claridad es una medida de la claridad del diamante, la profundidad es una medida de la profundidad del diamante, la mesa es una medida de la mesa del diamante, el precio es el precio del diamante, x, y y z son las tres dimensiones del diamante. Todas las variables son numéricas, excepto el corte, el color y la claridad, que son categóricas.

El objetivo es construir un modelo de regresión que pueda predecir el precio de venta de un diamante.

Ten en cuenta que para el uso de la funcion create_model y tune_model es recomendable tener instalado xgboost y/o LightGBM.

```
# importar las librerias necesarias
from pycaret.datasets import get_data
from pycaret.regression import *

# cargar el dataset de diamond
dataset = get_data('diamond')

# guardar algunos datos para probar cómo predice el modelo
data = dataset.sample(frac=0.9, random_state=786).reset_index(drop=True)
data_unseen = dataset.drop(data.index).reset_index(drop=True)

# inicializar PyCaret y especificar la columna objetivo
exp_reg101 = setup(data = data, target = 'Price', session_id=123)

# podemos comparar modelos para quedarnos con el mejor; por defecto
# el mejor se obtiene con una validación cruzada de 10 pliegues (k-fold, k=10:
# podemos usar otra medida a conveniencia, F1, Accuracy...
compare_models()

# crear uno o varios modelos de clasificación a partir de los mejores
# del paso anterior

#AdaBoost Regressor
ada = create_model('ada')
#trained model object is stored in the variable 'ada'.
print(ada)

#Light Gradient Boosting Machine
lightgbm = create_model('lightgbm')

#Decision Tree
```

```
dt = create_model('dt')

# mejorar el rendimiento de los modelos
tuned_ada = tune_model('ada')
print(tuned_ada)

tuned_lightgbm = tune_model('lightgbm')
print(tuned_lightgbm)

tuned_dt = tune_model('dt')
print(tuned_dt)

#trazar gráficamente los modelos
#gráfica de residuos (diferencias valor real - valor predicho)
plot_model(tuned_lightgbm)

# gráfica de errores de predicción
plot_model(tuned_lightgbm, plot = 'error')
#gráfica de la importancia de las características a la hora de predecir
plot_model(tuned_lightgbm, plot='feature')

# evaluar el rendimiento del modelo
evaluate_model(tuned_lightgbm)

# hacer predicciones en un conjunto de datos de prueba
predictions = predict_model(tuned_lightgbm, data=data)

#terminar el modelo para el despliegue
final_lightgbm = finalize_model(tuned_lightgbm)
print(final_lightgbm)
predict_model(final_lightgbm)
#predecir con datos no vistos antes por el modelo
unseen_predictions = predict_model(final_lightgbm, data=data_unseen)
unseen_predictions.head()
# la columna Label es la predicción

# guardar el modelo entrenado en disco
save_model(tuned_lightgbm, 'tuned_lightgbm_model')
#cargar el modelo para su uso
saved_final_lightgbm = load_model('tuned_lightgbm_model')
new_prediction = predict_model(saved_final_lightgbm, data=data_unseen)
new_prediction.head()
```

2.3.1 Más ejemplos

Hay un ejemplo más completo en GitHubm intermedio.