1_Pycaret_Heart

December 1, 2024

1 Preparación de datos:

[1]:
```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import joblib
import warnings

warnings.filterwarnings('ignore')
```

[2]:
```
# Cargar el dataset
file_path = "heart.csv"
df = pd.read_csv(file_path)
```

[3]:
```
# Separar variables de entrada y salida
X = df.drop("output", axis=1)
y = df["output"]
```

[4]:
```
# Definir las columnas para escalar y codificar
numeric_features = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']
categorical_features = ['cp', 'thall', 'caa']

# Crear el preprocesador
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(drop='first'), categorical_features)
    ])

# Crear el pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('smote', SMOTE(random_state=42)),
```

```
    ('classifier', RandomForestClassifier())
])
```

[5]:
```
# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

2 Modelación

2.1 Modelacion sin PyCaret

[6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
```

2.2 Busqueda de hiperparametros

[7]:
```
# Definir la búsqueda de hiperparámetros
param_grid = {
    'classifier__n_estimators': [50, 100, 200],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5, 10],
}

# Realizar la búsqueda de hiperparámetros
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

[7]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
StandardScaler(),
                                                                          ['age',
'trtbps',
'chol',
'thalachh',
'oldpeak']),
                                                                         ('cat',
OneHotEncoder(drop='first'),
                                                                          ['cp',
'thall',
'caa'])])),
                                       ('smote', SMOTE(random_state=42)),
```

```
                                    ('classifier',
                                     RandomForestClassifier())]),
             param_grid={'classifier__max_depth': [None, 10, 20],
                         'classifier__min_samples_split': [2, 5, 10],
                         'classifier__n_estimators': [50, 100, 200]})
```

[8]:
```
# Mejor modelo
best_model = grid_search.best_estimator_
print("\nMejores hiperparámetros para Random Forest:")
print(grid_search.best_params_)
```

```
Mejores hiperparámetros para Random Forest:
{'classifier__max_depth': None, 'classifier__min_samples_split': 2,
'classifier__n_estimators': 50}
```

[9]:
```
# Evaluar el mejor modelo
y_pred_best = best_model.predict(X_test)
print("\nBest Model Accuracy: {:.2f}".format(accuracy_score(y_test,␣
 ↪y_pred_best)))
print("Classification Report for Best Model:\n")
print(classification_report(y_test, y_pred_best))
```

```
Best Model Accuracy: 0.82
Classification Report for Best Model:

              precision    recall  f1-score   support

           0       0.76      0.90      0.83        29
           1       0.89      0.75      0.81        32

    accuracy                           0.82        61
   macro avg       0.83      0.82      0.82        61
weighted avg       0.83      0.82      0.82        61
```

[10]:
```
# Modelos base
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier(),
}

# Entrenar y evaluar modelos
best_base_model = None
```

```python
best_accuracy = 0
best_model_name = ""

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\n{name} Accuracy: {accuracy:.2f}")
    print(f"Classification Report for {name}:\n")
    print(classification_report(y_test, y_pred))

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_base_model = model
        best_model_name = name
```

```
Logistic Regression Accuracy: 0.89
Classification Report for Logistic Regression:

              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61


Decision Tree Accuracy: 0.80
Classification Report for Decision Tree:

              precision    recall  f1-score   support

           0       0.74      0.90      0.81        29
           1       0.88      0.72      0.79        32

    accuracy                           0.80        61
   macro avg       0.81      0.81      0.80        61
weighted avg       0.82      0.80      0.80        61


Random Forest Accuracy: 0.84
Classification Report for Random Forest:

              precision    recall  f1-score   support
```

```
           0       0.83      0.83      0.83        29
           1       0.84      0.84      0.84        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61


SVM Accuracy: 0.70
Classification Report for SVM:

              precision    recall  f1-score   support

           0       0.79      0.52      0.62        29
           1       0.67      0.88      0.76        32

    accuracy                           0.70        61
   macro avg       0.73      0.70      0.69        61
weighted avg       0.73      0.70      0.69        61


KNN Accuracy: 0.69
Classification Report for KNN:

              precision    recall  f1-score   support

           0       0.69      0.62      0.65        29
           1       0.69      0.75      0.72        32

    accuracy                           0.69        61
   macro avg       0.69      0.69      0.69        61
weighted avg       0.69      0.69      0.69        61
```

[11]:

```python
print("\nBest Base Model Accuracy: {:.2f}".format(best_accuracy))
print("Classification Report for Best Base Model:\n")
y_pred_best_base = best_base_model.predict(X_test)
print(classification_report(y_test, y_pred_best_base))
print(f"\nBest Base Model Name: {best_model_name}")
```

```
Best Base Model Accuracy: 0.89
Classification Report for Best Base Model:

              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32
```

```
    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

```
Best Base Model Name: Logistic Regression
```

[12]:
```
# Guardar el mejor modelo
joblib.dump(best_model, "mejor_modelo_pipeline.joblib")
```

[12]:
```
['mejor_modelo_pipeline.joblib']
```

2.3 Modelacion con PyCaret

[13]:
```
#!pip install pycaret
from pycaret.classification import *
```

[14]:
```
# Cargar el dataset
file_path = "heart.csv"
data = pd.read_csv(file_path)
```

[15]:
```
# Configuración inicial
clf_setup = setup(data=data, target="output",
                  train_size=0.8,  # 80% para entrenamiento
                  normalize=True,  # Normalizar datos
                  session_id=42,  # Reproducibilidad
                  fold=5)  # Validación cruzada
```

```
<pandas.io.formats.style.Styler at 0x12c380413f0>
```

[16]:
```
# Comparar y seleccionar los 5 mejores modelos
top_models = compare_models(n_select=5)

# Optimizar los 5 modelos seleccionados
tuned_models = [tune_model(model) for model in top_models]
```

```
Initiated  . . . . . . . . . . . . . . . . . .              08:21:10
Status     . . . . . . . . . . . . . . . . . .  Loading Dependencies
Estimator  . . . . . . . . . . . . . . . . . .      Compiling Library
```

```
<IPython.core.display.HTML object>
```

```
<pandas.io.formats.style.Styler at 0x12c380435b0>
```

```
Initiated  . . . . . . . . . . . . . . . . . .              08:21:17
```

```
Status     . . . . . . . . . . . . . . . . . . Loading Dependencies
Estimator  . . . . . . . . . . . . . . . . . .    Compiling Library
```

<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x12c360ea470>

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Original model was better than the tuned model, hence it will be returned. NOTE:
The display metrics are for the tuned model (not the original one).
```

```
Initiated  . . . . . . . . . . . . . . . . . .             08:21:19
Status     . . . . . . . . . . . . . . . . . . Loading Dependencies
Estimator  . . . . . . . . . . . . . . . . . .    Compiling Library
```

<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x12c3822ed40>

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
Initiated  . . . . . . . . . . . . . . . . . .             08:21:20
Status     . . . . . . . . . . . . . . . . . . Loading Dependencies
Estimator  . . . . . . . . . . . . . . . . . .    Compiling Library
```

<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x12c37e6a800>

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
Initiated  . . . . . . . . . . . . . . . . . .             08:21:20
Status     . . . . . . . . . . . . . . . . . . Loading Dependencies
Estimator  . . . . . . . . . . . . . . . . . .    Compiling Library
```

<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x12c3822e530>

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
Initiated  . . . . . . . . . . . . . . . . . .             08:21:21
Status     . . . . . . . . . . . . . . . . . . Loading Dependencies
Estimator  . . . . . . . . . . . . . . . . . .    Compiling Library
```

<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x12c3835a6b0>

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[17]:
```
# Evaluar cada modelo optimizado y seleccionar el mejor
best_model = None
best_combined_score = 0

for tuned_model in tuned_models:
    evaluate_model(tuned_model)
    metrics = pull().iloc[0]
    combined_score = (metrics['F1'] + metrics['Accuracy'] + metrics['Recall']) /
 ↪ 3

    if combined_score > best_combined_score:
        best_combined_score = combined_score
        best_model = tuned_model
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
 ↪options=(('Pipeline Plot', 'pipelin…

interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
 ↪options=(('Pipeline Plot', 'pipelin…

interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
 ↪options=(('Pipeline Plot', 'pipelin…

interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
 ↪options=(('Pipeline Plot', 'pipelin…

interactive(children=(ToggleButtons(description='Plot Type:', icons=('',),␣
 ↪options=(('Pipeline Plot', 'pipelin…
```

[18]:
```
best_model
```

[18]:
```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     monotonic_cst=None, n_estimators=100, n_jobs=-1,
                     oob_score=False, random_state=42, verbose=0,
                     warm_start=False)
```

[19]:
```
best_combined_score
```

[19]:
```
0.8348666666666666
```

[20]:
```
# Entrenar el modelo final con todos los datos
final_model = finalize_model(best_model)  # Selecciona el mejor modelo␣
 ↪optimizado
```

[21]:
```python
from pycaret.classification import save_model

# Guardar el modelo
save_model(final_model, "mejor_modelo_pipeline")
```

```
Transformation Pipeline and Model Successfully Saved
```

[21]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['age', 'sex', 'cp', 'trtbps',
                                              'chol', 'fbs', 'restecg',
                                              'thalachh', 'exng', 'oldpeak',
                                              'slp', 'caa', 'thall'],
transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
keep_empty_features=False,
missing_values=nan,
strategy='mean'))),
                 ('categor…
                  ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0,
                                       class_weight=None, criterion='gini',
                                       max_depth=None, max_features='sqrt',
                                       max_leaf_nodes=None, max_samples=None,
                                       min_impurity_decrease=0.0,
                                       min_samples_leaf=1, min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       monotonic_cst=None, n_estimators=100,
                                       n_jobs=-1, oob_score=False,
                                       random_state=42, verbose=0,
                                       warm_start=False))],
          verbose=False),
 'mejor_modelo_pipeline.pkl')
```

Aunque de los modelos con scikit-learn el mejor fue el modelo 'Logistic Regression', con PyCaret el mejor fue el ExtraTreesClassifier.